

# **EMULATION AND IMITATION VIA PERCEPTUAL GOAL SPECIFICATIONS**

A Dissertation  
Presented to  
The Academic Faculty

by

Ashley D. Edwards

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in the  
School of Interactive Computing

Georgia Institute of Technology  
May 2019

Copyright © 2019 by Ashley D. Edwards

# EMULATION AND IMITATION VIA PERCEPTUAL GOAL SPECIFICATIONS

Approved by:

Professor Charles L. Isbell, Advisor  
School of Interactive Computing  
*Georgia Institute of Technology*

Professor Tucker Balch  
School of Interactive Computing  
*Georgia Institute of Technology*

Professor Sonia Chernova  
School of Interactive Computing  
*Georgia Institute of Technology*

Professor Mark Riedl  
School of Interactive Computing  
*Georgia Institute of Technology*

Professor Pieter Abbeel  
Department of Electrical Engineering and  
Computer Sciences  
*University of California, Berkeley*

Date Approved: March 21, 2019

*For Tremayne Blackwell.*

## ACKNOWLEDGEMENTS

I have many people to thank for supporting and encouraging me during my time at Georgia Tech. First, I would like to thank my advisor, Charles Isbell, not only for the guidance in writing this dissertation, but also for always allowing me to pursue any research direction that I was interested in. Charles has been an incredibly supportive advisor, and because of this, my PhD has been an enjoyable experience.

I'd like to thank my thesis committee, Tucker Balch, Sonia Chernova, Mark Riedl, and Pieter Abbeel, for taking the time to provide insights and comments on my thesis.

I am lucky to have spent my PhD close to home. My family was always there when I needed them. To my parents, Michael and Phyllis, whose love and support has given me the strength to push through. Pa and Momma, thank you for your wisdom, your words of encouragement, and for “watching” Archie (my dog) “every now and then”.

To my brother and sister, Kita and Mook, who have been my number one squad throughout this all, and who are always willing to binge watch reality tv shows with me when I need to drain my brain a little. I am not ashamed.

And to my big brothers Mick and Chris, who are a bit further away, but always make sure to tell me that they are proud and to encourage me to stay focused. My favorite piece of advise is “Ash, the parties will always be there.” It really helps during deadline time!

To my boyfriend, Himanshu Sahni, who has been with me since day one, through all the highs and lows that is the PhD. Thank you for reminding me to remain a person and to enjoy life. And for helping me with so many experiments and papers and talks, etc. And thank you to Mummy, Papa, Vineet, and Ankita, for extending their arms whenever I am away from home.

To my best friends from the UGA, Shenika Lee, Vanessa Echeverria, Andrew J. Nelson,



and Dustin Beck, who were there even when I disappeared for months and joined the enemy, and now friend, that is Georgia Tech. You guys have been my champions and it really makes a difference to know that I have friends to come back to during the week in the year when I don't have a paper deadline.

To my friends I've met while at Georgia Tech, Shray Bansal, Adam McLaughlin, Ashwath Kumar, Daniel Kohlsdorf, Sasha Lambert, Steven Hickson, Amir Shaban, Safoora Yousefi, Xavier Collier, Kalesha Bullard, Saurabh Kumar, Qi Zhang, and Martin Levihn, for reminding me that there is life during grad school after all, and for spending time playing board games, camping, doing internships together, traveling, and just being people!

To my labmates, a.k.a. my pfunk family, that have acted as good friends and mentors during my PhD: Yannick Schroecker, Pushkar Kolhe, Jon Scholz, Luisca Cobo, Arya Irani, Liam Mac Dermid, Peng Zang, Chris Simpkins, Rahul Sawhney, and Kaushik Subramanian.

I have been lucky to intern and do research abroad within great teams. Thank you to my teams at Google Photos and Google Brain for introducing me to exciting industry research. And thank you to Professor Atsuo Takanishi for allowing me to do research at your lab at Waseda University in Japan.

I began my research in reinforcement learning as an undergrad at UGA, and I was lucky to have excellent mentors. Thank you to Andy Barto for accepting me into the lab at UMass for a summer as a young undergrad who had never even heard of reinforcement learning, and to George Konidaris for being my graduate mentor. Without this introduction I may not have even ended up in this field!

To William Pottenger, who also accepted me into his lab during one summer as an undergraduate. Your guidance and undying support and encouragement motivated me to write and present my first paper, and to confidently enter grad school.

I've dedicated my dissertation to my oldest brother, Tremayne, who passed away before I reached graduate school. Mayne was the most loving, care-free, and supportive brother one could hope for. And his words of encouragement, laughter, and spirit have been with

me throughout this entire journey, especially during the most difficult times. I thank and love you, brother.

Many people have made my time at Georgia Tech truly memorable. To anyone else that I may have forgotten to mention, thank you too!

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>iv</b>
<b>LIST OF TABLES</b>	<b>xii</b>
<b>LIST OF FIGURES</b>	<b>xiv</b>
<b>SUMMARY</b>	<b>.xviii</b>
<b>I INTRODUCTION</b>	<b>1</b>
1.1 Goal Specifications	1
1.2 Learning from alternative sources	2
1.3 Thesis Contributions	3
1.3.1 Outline of thesis claims	3
1.3.2 Specific contributions	4
<b>II BACKGROUND</b>	<b>6</b>
2.1 Machine Learning	6
2.1.1 Formalities	6
2.1.2 Supervised Learning	7
2.1.3 Unsupervised Learning	10
2.1.4 Reinforcement Learning	11
2.2 Deep Learning	11

2.2.1	Formalities . . . . .	12
2.2.2	Types of neural networks . . . . .	12
2.2.3	Activation functions . . . . .	13
2.3	Reinforcement learning . . . . .	13
2.3.1	Markov Decision Processes . . . . .	14
2.3.2	Bellman Equation . . . . .	15
2.3.3	Value iteration . . . . .	16
2.3.4	Monte-Carlo Approaches . . . . .	17
2.3.5	Q-Learning . . . . .	17
2.3.6	Exploration vs exploitation . . . . .	20
2.3.7	Reward specifications . . . . .	21
2.4	Imitation Learning . . . . .	21
2.5	Computer Vision . . . . .	22
2.5.1	Motion templates . . . . .	22
2.5.2	HOG features . . . . .	23
<b>III</b>	<b>RELATED WORK . . . . .</b>	<b>25</b>
3.1	Goal specifications for reinforcement learning . . . . .	25
3.2	Explicit goal instantiations . . . . .	26
3.3	Imitation Learning . . . . .	29

3.3.1	Classic approaches . . . . .	29
3.3.2	Correspondence problems . . . . .	30
3.3.3	Direct policy optimization methods . . . . .	30
3.3.4	Learning from state observations . . . . .	31
3.3.5	Multi-modal predictions . . . . .	31
<b>IV</b>	<b>PERCEPTUAL REWARD FUNCTIONS . . . . .</b>	<b>33</b>
4.1	Introduction . . . . .	33
4.2	Approach . . . . .	34
4.2.1	Formalities . . . . .	35
4.2.2	Perceptual reward functions . . . . .	36
4.2.3	Perceptual task descriptors . . . . .	37
4.3	Experiments and results . . . . .	39
4.3.1	Kobian Environment . . . . .	39
4.3.2	Results . . . . .	41
4.4	Discussion and conclusion . . . . .	44
<b>V</b>	<b>CROSS-DOMAIN PERCEPTUAL REWARD FUNCTIONS . . . . .</b>	<b>46</b>
5.1	Introduction . . . . .	47
5.2	Approach . . . . .	48
5.3	Formalities . . . . .	48

5.4	Cross-domain perceptual reward functions . . . . .	49
5.4.1	Network representation . . . . .	49
5.4.2	Training mechanism . . . . .	51
5.5	Experiments and results . . . . .	51
5.5.1	Maze environment . . . . .	53
5.5.2	Results . . . . .	53
5.6	Conclusion . . . . .	55
<b>VI</b>	<b>PERCEPTUAL VALUES FROM OBSERVATION . . . . .</b>	<b>57</b>
6.1	Introduction . . . . .	58
6.2	Formalities . . . . .	59
6.3	Approach . . . . .	60
6.3.1	Step 1: Learning values from observation . . . . .	61
6.3.2	Step 2: Learning action-values from values . . . . .	62
6.4	Experiments . . . . .	63
6.4.1	Results . . . . .	67
6.5	Discussion and Conclusion . . . . .	69
<b>VII</b>	<b>IMITATING LATENT POLICIES FROM OBSERVATION . . . . .</b>	<b>70</b>
7.1	Introduction . . . . .	70
7.2	Approach . . . . .	72

7.2.1	Problem formulation . . . . .	72
7.2.2	Step 1: Learning latent policies . . . . .	74
7.2.3	Step 2: Action Remapping . . . . .	77
7.3	Experiments and results . . . . .	80
7.3.1	Classic control environments . . . . .	81
7.3.2	Cartpole . . . . .	82
7.3.3	Acrobot . . . . .	82
7.3.4	Results . . . . .	83
7.3.5	CoinRun . . . . .	83
7.3.6	Results . . . . .	84
7.4	Discussion and conclusion . . . . .	85
<b>VIII</b>	<b>DISCUSSION AND FUTURE WORK . . . . .</b>	<b>87</b>
8.1	Introduction . . . . .	87
8.2	Perceptual rewards . . . . .	87
8.3	Imitation from other agents . . . . .	87
8.4	Safety . . . . .	88
<b>IX</b>	<b>CONCLUSION . . . . .</b>	<b>89</b>
9.1	Strengths and limitations . . . . .	89
9.1.1	Goal specification . . . . .	89

9.1.2	Reward type . . . . .	90
9.1.3	Generalization . . . . .	90
9.1.4	Action type . . . . .	90
9.1.5	Learning mechanism . . . . .	91
9.2	Performance against traditional reward design . . . . .	91
9.3	Extending beyond goal-directed tasks . . . . .	91
9.3.1	Options and constraints for perceptual emulation . . . . .	92
9.3.2	Options and constraints for perceptual imitation . . . . .	93
<b>Appendix A . . . . .</b>		<b>94</b>
<b>REFERENCES . . . . .</b>		<b>99</b>



## LIST OF TABLES

1	Comparison between the methods described in this dissertation: PRFs, CDPRs, PVO, and ILPO. . . . .	89
---	--	----

## LIST OF FIGURES

1	Examples of machine learning models. Note that these graphs are not based on any real data. Linear regression (left) aims to find a model that best fits the data, whereas logistic regression (right) aims to find a model that best separates different classes. . . . .	8
2	Example of the sigmoid or logistic function. When the inputs to this function are large, the output approaches 1. When the inputs are small, the outputs approach 0. . . . .	9
3	Example of a fully-connected deep neural network. . . . .	11
4	Example of the reinforcement learning setup. An agent takes actions in some state and receives rewards from the environment. . . . .	13
5	The image on the left shows the result of an agent moving from the <b>start</b> location to the top-right position. The middle image is the corresponding motion template. The image on the right is a zoomed-in visualization of the HOG features of the motion template. . . . .	22
6	The Kobian environment. In the Kobian Simulator, the agent must move parts of its face to make expressions. . . . .	39
7	Action Units (AUs) applied to Kobian’s face. The top left shows Kobian’s neutral face. The remaining top row images represent actions: Inner Brow Raiser, Outer Brow Raiser, Brow Lowerer, Upper Lid Raiser, Upper Lip Raiser and Lip Corner Puller. The bottom row represents actions: Lip Corner Depressor, Chin Raiser, Lip Stretcher, Lip Tightener, Lip Pressor, Lips Part, Jaw Drop, Mouth Stretch. Additionally, the agent can take the No-Op action $N$ . Note that these are the actions applied to Kobian’s neutral face. Each action can be applied on top of another to obtain faces not shown here. . . . .	40
8	Goal representations for Kobian. The faces of Kobian are the desired goals for the standard reward function, which is based on the internal variables of the agent. The human faces (©Jeffrey Cohn) are the desired goal for the face PRF, and the motion templates below are the desired goals for the motion PRF. . . . .	41
9	The rewards obtained in Kobian for each type of reward function. We ran the Kobian simulator for 2500 episodes for each experiment and averaged over 3 runs. We plot each graph separately because the rewards occur on different scales. . . . .	42

10	Sequence of highest rewarded “happiness” faces for standard (top), motion (middle) and face (bottom) reward functions. . . . .	42
11	Sequence of highest rewarded “surprise” faces for standard (top), motion (middle) and face (bottom) reward functions. . . . .	43
12	The distance $D$ between the HPRF goal template, $T_G$ , and the motion templates computed during each episode of the VRF, KPRF, and HPRF happiness and surprise tasks. . . . .	44
13	An example of how a reinforcement learning agent (left) typically learns tasks, versus a human (right). Agents typically have goals that are specified in its own configuration space, whereas humans can learn from many different task representations. . . . .	46
14	Deep neural network used for learning Cross-Domain Perceptual Reward (CDPR) functions. Two separate, though identical, networks encode features for the agent states $G_i$ and cross-domain goals $\hat{G}_i$ , respectively. Each network consists of a set of convolutional layers followed by a set of fully-connected layers. The exact components of each layer are dependent on the problem. The dot product between the outputs of each encoding layer represents the CDPR, $R(G_i, \hat{G}_j)$ . . . . .	50
15	Environment used in RL experiments. In the Maze domain, the agent’s task is to navigate to a specific room. The goal specifies the room the agent needs to navigate to. For the cross-domain goals, we used a sign language handshape of the first letter of the desired room’s color, and a spectrogram of a spoken command, “Go to the [color] room.” . The leftmost column represents the intra-domain goal representations, while the remaining two are the cross-domain goal representations. The top and bottom rows are referred to as Goal 1 and Goal 2 in our experiments. . . . .	52
16	Qualitative results in the Maze domain. To obtain these results, we spawned the agent in each location of the maze, and then produced a heat map of the rewards received. The results are best seen in color. . . . .	54
17	Smoothed results for running RL in the Maze domain. We ran RL with the goal specifications described in Figure 15a. . . . .	55
18	Emulation vs Imitation. There is a trade-off between the amount of information we give the agent in a single demonstration and the amount of experience it will take the agent to learn. Imitation by Observation then offers a middle-ground between environment experience and demonstration data. . . . .	57
19	Value assignment for a length $T$ trajectory sampled from expert demonstrations. . . . .	60

20	Heatmap of values learned by PVO in unseen maze environments. Brighter colors have larger values. The results are best seen in color. . . . .	64
21	Values learned by PVO in the pouring dataset. The top row represents frames from a single, unseen video. The bottom row represents the learned values for each frame in the video. . . . .	65
22	Example of a procedurally generated levels in CoinRun. In this environment, an agent needs to reach a single coin at the end of a platform. . . . .	66
23	CoinRun results. The trials were averaged over 5 runs with a different, unseen, procedurally generated level for each method. The policy was evaluated for 10 runs every 10 steps. The evaluation metric was a negative step cost. . . . .	67
24	The latent policy network learns a latent policy, $\pi(z s)$ , and a forward dynamics model, $G$ . . . . .	74
25	The action remapping network learns $\pi(a s_t, z)$ to align the latent actions $z$ with ground-truth actions $a$ . . . . .	77
26	Cartpole and Acrobot results. The trials were averaged over 50 runs for ILPO and the policy was evaluated every 50 steps. The reward used for training the expert and evaluation was +1 for every step that the pole was upright in cartpole, and a -1 step cost for acrobot. . . . .	79
27	Cartpole and Acrobot results for selecting $ Z $ . The trials were averaged over 5 runs for ILPO and the policy was evaluated every 50 steps and averaged out of 10 policy runs. The reward used for training the expert and evaluation was +1 for every step that the pole was upright in cartpole, and a -1 step cost for acrobot. . . . .	80
28	CoinRun environment used in experiments. CoinRun consists of procedurally generated levels and the goal is to get a single coin at the end of a platform. We used an easy level (left) and hard level (middle and right) in our experiments. The middle image is the agent's state observation in the hard level and the image on the right is a zoomed out version of the environment. This task is difficult because the gap in the middle of the platform can not be recovered from and there is a trap as soon as the agent reaches the other side. When training, the images also include a block showing x and y velocities. . . . .	81
29	CoinRun imitation learning results. The trials were averaged over 50 runs for ILPO and BCO the policy was evaluated every 200 steps and averaged out of 10 policy runs. The reward used for training the expert and evaluation was +10 after reaching the coin. . . . .	82

- 30 Next state predictions computed by ILPO in the CoinRun easy task. The highlighted state represents the closest next state obtained from equation 44. 83

## SUMMARY

This dissertation aims to demonstrate how perceptual goal specifications may be used as alternative representations for specifying domain-specific reward functions for reinforcement learning. The works outlined in this document aim to validate the following thesis statement: **Employing perceptual goal specifications for goal-directed tasks: is as straightforward as specifying domain-specific rewards; is a more general representation for tasks; and equally enables task completion.** We describe various approaches for specifying goals visually and how we may compute rewards and learn policies directly from these representations:

- Chapter 4 introduces Perceptual Reward Functions and describes how we can utilize a hand-defined similarity metric to enable learning from goals that are different from an agent's.
- Chapter 5 introduces Cross-Domain Perceptual Reward Functions and describes how we can learn a reward function for cross-domain goal specifications.
- Chapter 6 introduces Perceptual Value Functions and describes how we can learn a value function from sequences of expert observations without access to ground-truth actions.
- Chapter 7 introduces Latent Policy Networks and describes how we can learn a policy from sequences of expert observations without access to ground-truth actions.

The remaining chapters motivate and provide background for this dissertation and outline a plan for future research.

# Chapter I

## INTRODUCTION

Reinforcement Learning (RL) is an approach for training artificial agents to make sequential decisions in the world. Much of the power behind RL is that we can use a single signal, known as the reward, to indicate desired behavior. Rewards can be surprisingly powerful and can motivate agents to solve problems without any further guidance for how to accomplish them. However, defining these rewards can often be difficult. This dissertation introduces an alternative to the typical reward design mechanism. In particular, we introduce methods that allow one to focus on specifying goals, rather than scalar rewards.

### *1.1 Goal Specifications*

In RL, the reward signal is often described as being received from the agent’s environment as it takes actions in the world and encounters both positive and negative situations. However, in practice, the reward is not a natural signal that can be obtained automatically. Rather, it must typically be engineered by a human designer for each task. One problem with this type of formulation is that it requires knowledge of the agent’s environment and internal variables, for example the agent’s location and other objects in the world. In addition to this, reward functions are often not general, and need to be specified each time the task changes.

In traditional RL settings, the reward function has been inherently domain-specific. This representation has led to many successes in difficult domains, but it is often not straightforward to develop. For example, consider a robot that has a task of building origami figures. Designing a reward function for this problem would require defining some notion of correctness for *each* desirable figure. Furthermore, it is unclear what the inputs to the reward function should even be. Essentially, designing the reward for complex tasks often requires a detailed understanding of the agent’s internal configuration and how to construct

a goal in relation to it.

In order to build agents that can learn from novel situations, we should aspire to develop more general representations for goals that can be more easily specified and have the ability to be reused. Furthermore, we should consider learning from situations that are not only based on the agent’s environment, but flexible enough to be agnostic to certain components of the domain.

## ***1.2 Learning from alternative sources***

One alternative to defining domain-specific rewards is to provide examples of the desired behavior. For instance, imitation learning approaches allow people to provide demonstrations of the task. However, such approaches have traditionally required solving the problem within the agent’s environment.

When people solve problems, we regularly exploit alternative learning materials *outside* of the configuration of a task, be it through diagrams, videos, text, and speech. For example, if we were to build an origami crane, we might look at an image of a completed figure to determine if our own model is correct. This dissertation will describe similar approaches for presenting tasks to agents. In particular, I will introduce methods for specifying *perceptual* goals both within and outside of an agent’s environment, and perceptual reward functions, values, and policies that are derived from these goals. The approaches in this document aim to allow us to represent goals in settings where we can more easily find or construct solutions, without requiring us to modify the reward function when the task changes. We will focus on finding solutions to goal-directed tasks. That is, problems with an explicitly specified goal that, when reached, terminates the task. This differs from a continuing task, such as driving a car, which does not necessarily have a specific terminating goal-condition.

By removing domain-specific aspects of the problem, we demonstrate that goals can be expressed while being agnostic to the reward function, action-space, or state-space of the agent’s environment.



### 1.3 Thesis Contributions

The thesis of this dissertation is:

Employing perceptual goal specifications for goal-directed tasks: is as straightforward as specifying domain-specific rewards; is a more general representation for tasks; and equally enables task completion.

This section outlines how I will demonstrate each claim in my thesis.

#### 1.3.1 Outline of thesis claims

1. Perceptual goals are as straightforward to specify as domain-specific rewards.
  - (a) We demonstrate that we can represent goals through images rather than domain-specific reward functions.
  - (b) We demonstrate that these images can come from sources *we choose*, rather than the agent’s environment.
  - (c) We demonstrate that these representations can be agnostic to certain parts of the agent’s environment.
2. Perceptual goals are a more general representation for tasks.
  - (a) We demonstrate that perceptual-based reward functions only need to be specified once and do not need to be modified when the task or goal changes.
  - (b) We demonstrate that we can use the same learned perceptual-based reward functions for unseen tasks.
  - (c) We demonstrate that learned perceptual values can be used for unseen tasks.
  - (d) We demonstrate how perceptual goals can be used specifying rewards.
3. Perceptual goals equally enable task completion.

- (a) We demonstrate that perceptual goals used in reinforcement learning can perform as well as and outperform standard agents.
- (b) We demonstrate that perceptual goals used in imitation learning can perform as well as and outperform standard agents.

### 1.3.2 Specific contributions

This section summarizes the approaches that will be introduced in this dissertation. We are interested in learning when part of the domain is unknown: the actions available to the agent, rewards, or environmental setting. The approaches described in this dissertation utilize a blend of reinforcement and imitation learning to avoid developing domain-specific reward functions. Each of the contributed methods will use images to specify goals, either as a single observation or a sequence obtained from an expert. Each method that we describe will be agnostic to the agent’s internal reward and learn from observations only, and so will additionally be agnostic to the actions.

In Chapter 4, we will introduce a hand-defined correspondence for representing Perceptual Reward Functions (PRFs) [26]. We will describe an approach for representing goals through motion and a general method for specifying the reward function. This approach will allow specifying a goal that was obtained from a chosen source outside of the agent’s environment. As such, it is agnostic to the agent’s state representation. The reward function remains unchanged when the task changes which demonstrates generality. We will evaluate this method in a simulated robotic environment and show that it performs similarly to a standard reinforcement learning agent.

Hand-defined PRFs are useful because they only utilize a single goal sample, but this approach requires manual tuning. Chapter 5 will describe an approach for learning PRFs for goals specified in environments that are different than the agent’s [28]. In particular, we use deep learning to learn cross-domain similarities between goals specified outside the agent’s environment and agent states. As such, this approach is also agnostic to the agent’s

state representation. Additionally, the reward function remains unchanged and can be used for unseen tasks, hence demonstrating generality. We will evaluate this method in a maze environment with two different goal specifications and show that it performs similarly to a standard reinforcement learning agent.

The previous approach is useful as it allows one to represent goals in surrogate environments, but it requires samples of goals in both the agent and cross-domain environments. It may be useful to infer the goal from experiences. We will describe two methods for imitating directly from observations.

Chapter 6 will describe how we can learn values from observation. We are interested in learning a general value function from observations. We use this model to specify values in unseen environments, hence demonstrating generality. We will evaluate this method in a platform game and maze environment.

The previous methods still required learning a reward function for reinforcement learning. Chapter 7 will describe how we can learn policies directly from observation without specifying rewards at all [29]. We will evaluate in classic control environments and a platform game.

## Chapter II

### BACKGROUND

The focus of this dissertation is to develop perceptual goal specifications for imitation and reinforcement learning problems. The approaches that will be introduced in later chapters each involve learning from image inputs, and so it is necessary to use techniques that can handle visual representations. As such, in this chapter, we provide a brief overview of relevant machine learning, deep learning, reinforcement learning, and computer vision topics.

#### ***2.1 Machine Learning***

The world consists of an abundance of information and data that we use to make inferences from. When we look outside, we can often determine if it is going to rain or not. Or after falling down, we may figure out which steps to avoid next time. Notably, we do not need to have seen these exact scenarios in order to draw these conclusions. Rather, we have the ability to use past experiences to extract patterns that we can use to make decisions about the world. Nevertheless, there is often too much data and too many problems for humans to effectively solve on their own. Machine learning enables making similar inferences that inform decisions, but can be done on a much larger scale for many tasks—and often more accurately.

##### **2.1.1 Formalities**

In machine learning, we are interested in approaches that analyze data to draw inferences and make decisions. For example, given a list of symptoms, we may wish to determine if a person is sick. Or given an image, we may wish to automatically detect who is in the picture.

More formally, machine learning approaches aim to learn from *samples* of experiences

represented by a random vector  $\mathbf{x}$ . This vector consists of *features*  $x_1, x_2, \dots, x_n \in \mathbf{x}$  that correspond to the relevant information for the problem. For instance, for the problem of determining if a person is sick, the features might be temperature, whether or not the person has a cough, or other typical questions the doctor might ask to make a diagnosis. Or for the problem of detecting people in a picture, the features might be the pixels in the image. We typically assume that these samples are *independent and identically distributed* (i.i.d.) which means that they are independently drawn from the same distribution [10].

One way to make inferences from these samples could be to simply store each sample in a database and extract it each time it was encountered again. This approach however would not be able to make any decisions about novel situations. Furthermore, it does not provide any understanding of the data itself. Given a list of samples, known as a *training set*, we often aim to make predictions about unseen samples. That is, we want to create systems that are capable of *generalizing* to new scenarios.

Machine learning approaches are typically divided into three areas: supervised, unsupervised, and reinforcement learning. We will briefly discuss each in the following sections.

### 2.1.2 Supervised Learning

In supervised learning, the training set consists of samples  $x$  paired with some *label*  $y$ . Labels indicate some ground-truth information about the sample, such as the diagnosis of a patient given their symptoms. The aim of supervised learning is to use these labels to do *function approximation*. Given  $x$ , we are interested in learning an estimate of the function  $f(x)$  that can predict the true output  $y$ . In particular, we aim to learn  $p(y|x)$ .

#### 2.1.2.1 Linear Regression

In regression tasks, labels are represented through real numbers. For example, given samples that contain information about the size of the house,  $y$  could be the house's price.

Figure 1 gives an example of a regression problem in supervised learning. In this

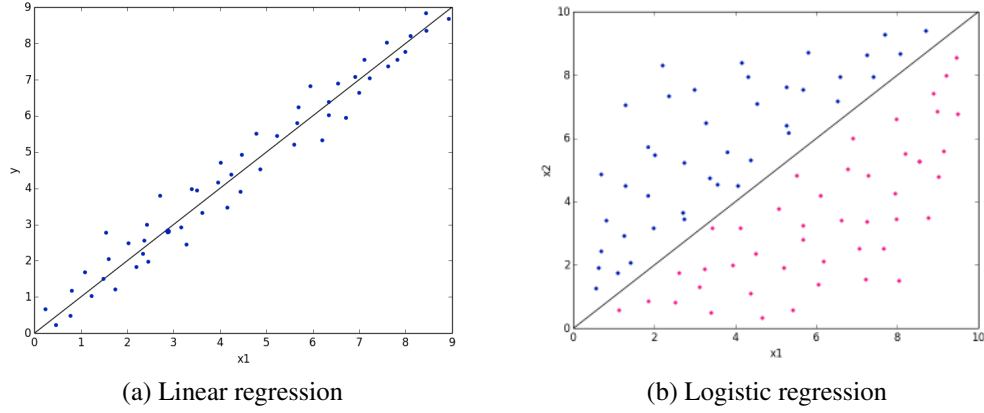


Figure 1: Examples of machine learning models. Note that these graphs are not based on any real data. Linear regression (left) aims to find a model that best fits the data, whereas logistic regression (right) aims to find a model that best separates different classes.

example, given a single feature  $x_1$ , the goal is to predict some output  $y$ .

Most of the supervised learning approaches that we consider in this dissertation will use some form of regression, although we typically assume the functions are non-linear.

Given training samples  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ , where  $n$  is the number of training samples, we aim to learn a function, also known as a hypothesis, that approximates the corresponding label  $y$ ,  $\hat{y}$ .

We typically use weights to formulate this hypothesis. In linear regression, we express the hypothesis as a linear combination of the features:

$$\hat{y} = \theta^T \mathbf{x}. \quad (1)$$

The purpose of the weights  $\theta$  is to determine the importance of each feature when predicting  $\hat{y}$ . We typically append a value of 1 to  $\mathbf{x}$  in order to learn a *bias* variable that shifts the prediction vertically.

Linear regression aims to find a line that best fits the data. It aims to minimize the difference between the output from the training set and the hypothesis. We will describe how to learn a model in a later section.

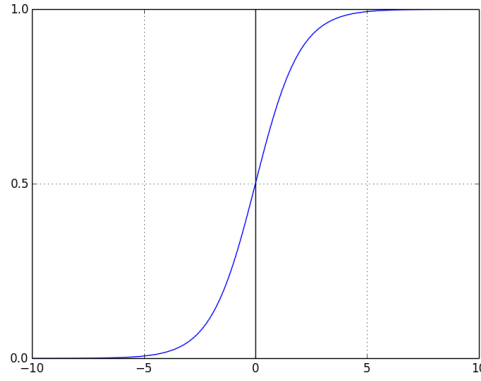


Figure 2: Example of the sigmoid or logistic function. When the inputs to this function are large, the output approaches 1. When the inputs are small, the outputs approach 0.

#### 2.1.2.2 Logistic regression

In logistic regression, the labels are known as classes or categories. The goal is to predict the probability of some variable  $x$  belonging to class  $y$ . Typically, classes are represented through a binary number that indicated whether some condition is true or false. Logistic regression aims to find a separating boundary between true and false samples.

When making categorical predictions we need to threshold the output so that it is between 0 and 1. We can formulate the hypothesis as:

$$H(\mathbf{x}) = \sigma(\theta^T \mathbf{x}). \quad (2)$$

Here,  $\sigma$  is known as the sigmoid or logistic function. It is defined as:

$$\sigma(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} \quad (3)$$

The sigmoid function is convenient because it squashes its inputs between 0 and 1, and hence is ideal when predicting probabilities. As  $\theta^T \mathbf{x}$  becomes larger than 0, the output gets closer to 1, as it becomes smaller than 0, the output gets closer to 0, and at 0 the output is .5. An example is shown in figure 2.

### 2.1.2.3 Gradient descent

When solving machine learning problems, there is typically a *loss function* that we aim to minimize. For example, in linear and logistic regression, the goal is to make the hypothesis  $\hat{y}$  be close to the labels  $y$ . A common approach for this would be to represent the loss as the Mean-Squared Error (MSE):

$$L_{\theta} = \frac{1}{n} \sum_i^n (\theta^T \mathbf{x}^{(i)} - y^i)^2, \quad (4)$$

where  $n$  is the number of training samples [33]. In order to learn this function, we use *gradient descent* to find parameters that minimize the loss. To do this, we take the gradient of the loss and use this gradient to update the weights:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla L_{\theta}. \quad (5)$$

Here,  $\alpha$  is a learning rate that determines how quickly the gradients are updated.

When making probabilistic predictions, like in logistic regression, another common loss function would be the cross-entropy loss [10]:

$$L_{\theta} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})). \quad (6)$$

Traditional gradient descent approaches required iterating over the entire batch of training data at once to make updates. Stochastic gradient descent enables learning from *minibatches* of data rather than training the entire dataset at once [13]. This is useful for learning when the dataset is large.

### 2.1.3 Unsupervised Learning

Unlike in supervised learning, unsupervised learning problems are not given access to labels. Rather, the aim of this approach is to learn structure from the data or to predict densities. For example, we can use this approach to cluster samples into groups that inform us of



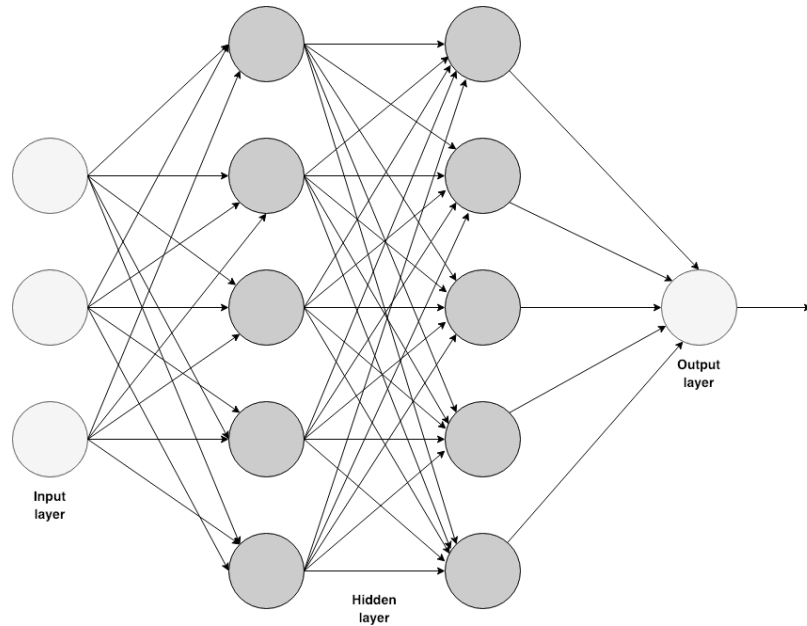


Figure 3: Example of a fully-connected deep neural network.

patterns about the data. We can also use unsupervised learning to train a generative model to generate data.

#### 2.1.4 Reinforcement Learning

In reinforcement learning, we aim to learn behaviors from signals known as rewards. We will discuss more of this approach in a later section.

## 2.2 *Deep Learning*

Deep learning is often used to train machine learning models, particularly those with large amounts of high-dimensional data such as images or audio. We use deep learning to learn features from images for reinforcement and imitation learning. Given that we aim to solve problems with images, it is invariably necessary to turn to deep learning, which has been shown to handle image inputs better than basic approaches. In this section, we explain some of the terminology that will be used in later chapters.

### 2.2.1 Formalities

Deep neural networks consist of *layers* of individual *units* that each compute a function for some given input. An example is shown in figure 3. The input layer consists of the raw data, such as pixels from images. Each hidden layer computes a function over the previous layers inputs. The output layer makes predictions given the computations made by the final layer.

Typically, the hidden units each compute the weighted sum of the inputs, like in linear regression. However, there are several advantages in deep learning. First, each function is placed through an activation unit that makes it nonlinear, and secondly, the hidden layers consist of the outputs of the previous ones, thus allowing the network to make abstractions.

Deep neural networks typically consist of many more parameters than other machine learning methods. An approach known as *backpropagation* is typically used to train these models [33].

### 2.2.2 Types of neural networks

In this section, we will briefly discuss the types of neural networks used in this dissertation.

#### 2.2.2.1 Feed-forward networks

In feedforward networks, information from the beginning input layer flows through each hidden layer into the output layer. In particular, the functions computed by the previous layer are used as inputs to the next layer. Many common types of neural networks are based off of this representation.

#### 2.2.2.2 Fully-connected models

In fully-connected networks, each of the outputs computed by the units in layer  $t$  is connected to every unit in layer  $t + 1$ . This type of network is illustrated in figure 3.

### 2.2.2.3 Convolutional neural networks

When using images as inputs, full-connected models are unlikely to capture the local dependencies between nearby pixels. Convolutional neural networks were introduced to better handle this representation. In this approach, groups of fixed-sized blocks, known as *filters*, slide across the image with a fixed increment, known as the *stride*. Each filter consists of weights, and the approach strings together the weighted sums of the filter weights with its current location in the image. This allows the network to find patterns across data.

### 2.2.3 Activation functions

In order to ensure that neural networks are capable of computing non-linear functions, each node in a hidden layer typically uses an *activation function* after computing the weighted sum. For example, the sigmoid function described in section 2.1.2.2 could be used. More commonly, a rectified linear unit (ReLU) is used [33], and is defined as:

$$g(z) = \max\{0, z\}. \quad (7)$$

## 2.3 Reinforcement learning

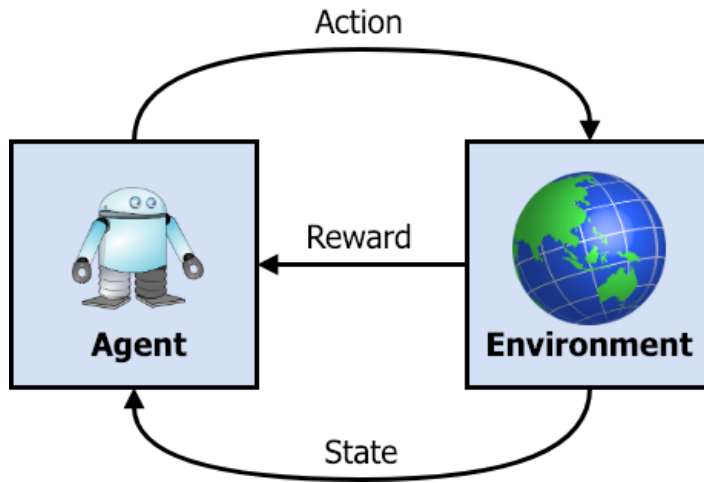


Figure 4: Example of the reinforcement learning setup. An agent takes actions in some state and receives rewards from the environment.

Often, we are interested in allowing artificial systems to take actions in the world to simplify our lives. For example, a robot could do the dishes for us after a long day at work or water our plants when we are out of town.

Representing these tasks with the previously described approaches could require a lot of effort. For example, providing labels for every action the robot should take in every setting would be intractable. But without any labels, the agent would have no understanding of the task at all.

Reinforcement learning (RL) provides a solution to this problem through a single signal—the reward. In particular, RL allows specifying rewards in desirable settings, and then allowing the system to learn which actions to take in order to maximize these rewards. The goal of reinforcement learning problems is to train an artificial system to maximize the long-term expected reward it receives over time.

### 2.3.1 Markov Decision Processes

Reinforcement learning problems are specified through a Markov Decision Process  $\langle S, A, P, R \rangle$  [83]:

- $S$  consists of states  $s \in S$  that represent the current variables of an agent's environment that it uses for learning.
- $A$  defines the agent's actions  $a \in A$  that it can take in its environment.
- $R$  represents the rewards  $r \in R$  that the agent receives for entering a state.
- $P$  specifies the agent's transition model  $P(s_{t+1} | s, a)$ .

In reinforcement learning, an *agent* interacts with its environment by taking actions and receives rewards for the actions it takes. This interaction is illustrated in figure 4. The goal of reinforcement learning is to learn a *policy*  $\pi(s) \rightarrow a$  that represents the probability of taking action  $a$  in state  $s$ , thus informing the agent of which actions to take in every state.

Importantly, Markovian means that all information that the agent needs to make decisions should be encapsulated in a single state and action. More formally, this means that only the previous state and action are considered when making decisions:

$$P(s_{t+1}, r_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \dots) = P(s_{t+1}, r_{t+1} | s_t, a_t). \quad (8)$$

In this dissertation, our focus will be on *episodic* tasks. In episodic tasks, the agent takes actions in the environment until it reaches some terminal state. Then, it is replaced in an initial start location where a new episode begins.

### 2.3.2 Bellman Equation

In reinforcement learning, we are interested maximizing the long-term expected return for a state, which can be defined as:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_k^T \gamma^k r_{k+t}, \quad (9)$$

where  $T$  may be a finite number or infinite. The discount factor  $0 \leq \gamma < 1$  encodes how rewards retain their value over time, and additionally stops the sum from becoming infinite. A small discount factor would mean that the agent was more *myopic*, and considered short-term goals. In practice, we often use larger values of the discount factor so that the agent can reach goals that are further out.

The reason we aim to learn the long-term reward, or value, of a state is because if an agent maximized short-term rewards, it might not learn about more interesting rewards that are further away.

The expected *value* of a state given some policy  $\pi$  can be specified as:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]. \quad (10)$$

This represents the long-term expected reward for some state  $s$ , given the current policy  $\pi$ .

Often, we are also interested in determining what the value is for taking some action  $a$  in state  $s$ . The expected *action-value* of a state given some policy  $\pi$  can be specified as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]. \quad (11)$$

We are typically interested in finding optimal values and action-values:

$$V^* = \max_\pi V_\pi(s), \forall_s \quad (12)$$

$$Q^* = \max_\pi Q_\pi(s, a), \forall_{s,a}. \quad (13)$$

This means that we aim to find the policy that maximizes the long-term expected reward. Obtaining a policy from Q-values is straightforward:

$$\pi(s) = \max_a Q(s, a). \quad (14)$$

We can iteratively approximate these values recursively using the Bellman equations [84].

$$V(s) := \max_a \sum_{s'} T(s, a, s') [r + \gamma V(s')] \quad (15)$$

$$Q(s, a) := \sum_{s'} T(s, a, s') [r + \max_{a'} Q(s', a')] \quad (16)$$

### 2.3.3 Value iteration

In this section, we will describe how we can learn values when we have a known transition and reward model. We can define this function recursively and use memoization to update the values. This section discusses a common dynamic programming method known as value iteration.

Value iteration approaches make updates to values by iterating over the entire state-space and computing values based on the current estimate. The update for value iteration is:

---

**Algorithm 1** Q-Learning

---

```
1: function LEARN(env)
2:   while learning do
3:     while not terminal do
4:        $a = \pi(s)$  ▷ Obtain action from policy
5:        $s', r, \text{terminal} = \text{env.step}(a)$  ▷ Take action and observe
6:        $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q'(s, a) - Q(s, a)]$  ▷ Update Q-values
7:        $s \leftarrow s'$ 
```

---

$$V(s) := \max_a \sum_{s'} T(s, a, s') [r + \gamma V(s')] \quad (17)$$

Notice that this update uses the current estimates of the values to make updates. This is known as *bootstrapping* [83].

### 2.3.4 Monte-Carlo Approaches

In general, we often do not have access to the transition model, and when we do, it is intractable to iterate over the entire state-space. Furthermore, most of the learning approaches that use in this dissertation are *model-free* and do not have access to  $P$ .

Reinforcement learning introduces several mechanisms for learning values without the model. One simple mechanism for learning them is to estimate values from samples of experiences. It is often more tractable to learn directly from the agent's interactions in the world. Monte-carlo approaches estimate action values simply by taking the average over samples.

### 2.3.5 Q-Learning

One problem with monte-carlo methods is that learning does not occur until the end of the episode. Temporal-difference TD methods mitigate this by using *bootstrapping* to estimate values. In this dissertation, we use a TD-method known as Q-learning, which learns action-values by making the following update rule:

$$Q(s, a) := Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)], \quad (18)$$

In words, this function updates  $Q(s, a)$  by moving it closer to  $r + \gamma Q(s', a')$ , which is the target that  $Q(s, a)$  should aim to approximate. Here,  $\alpha$  is a learning rate that, like in gradient descent, determines how much to change the current estimate.

When  $s'$  is a terminal state, then we make the update as:

$$Q(s, a) := Q(s, a) + \alpha[r - Q(s, a)], \quad (19)$$

This is because the agent cannot take actions in terminal states. Similarly, the action values for terminal states are typically assigned 0.

Q-learning offers a blend of monte-carlo and dynamic programming approaches. Like monte-carlo, it makes updates based on samples of experience and so it is model-free. Like dynamic-programming, it uses bootstrapping to make the updates and thus can learn during an episode instead of at the end.

Q-learning is known as an *off-policy* method because, rather than using the agent's current policy, it assumes that the agent is going to take the best action in state  $s'$ . This is a powerful assumption, because it allows using any policy, even a random one, to learn the values. This becomes important in later sections when we use batches of sampled data to update the values.

It is also possible to train action-values using *on-policy* methods that take into account the agent's policy:

$$Q(s, a) := Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)], \quad (20)$$

where  $a' \sim \pi(s)$  is the action the agent actually took in the environment.

This type a method is useful because sometimes the agent does not actually take the best action and so overestimate what the true values are. But we do not use these methods in this



dissertation and so they will not be discussed.

#### 2.3.5.1 *Tabular methods*

The simplest form of Q-learning allows action-values to be stored in a table. For example, if the state space were a gridworld then the action-value for taking action  $a$  in state  $\{x, y\}$  would be stored in  $\{x, y, a\}$ . Grids, often known as gridworlds in RL nomenclature, are the canonical example that most approaches should attempt to solve, before attempting more complicated environments.

#### 2.3.5.2 *Linear function approximation*

The state representations that work well for tabular Q-learning tend to be low dimensional. However, with large or continuous state spaces, learning from a table can be intractable. First, practically, storing such tables requires a lot of system memory. But even with infinite storage, reinforcement learning may still perform poorly with large or high-dimensional state spaces. This is because it is unlikely that the agent will ever reach the exact same state more than once. As such, we aim to utilize function approximation to interpolate across the state space. In linear function approximation, values can be approximated as:

$$Q_{\theta}(s) = \theta^T \phi_{s,a}. \quad (21)$$

Here,  $\theta$  are weights we are try to learn, and  $\phi$  is a function that extracts features from  $s$ . In order to train the weights, we aim to minimize the following loss:

$$L_{\theta} = \|y - Q_{\theta}(s, a)\|^2, \quad (22)$$

where  $y = r + \gamma \max_a Q(s', a')$ . Note that this mechanism for learning is very similar to tabular q-learning, in that we aim to move our current estimate closer to  $y$ . The main difference is that now  $Q$  is a function of  $\theta$ .

The weights indicate how important each feature is for estimating values. We can use

gradient descent to learn the weights as in the supervised learning methods described before.

#### 2.3.5.3 Deep Q-Learning

Often, value functions can not be explained from linear combinations only. Furthermore, it may be difficult to hand-craft features. In our case, the state inputs will be images and so we use a deep reinforcement learning method known as a Deep Q-Network (DQN) [54] to approximate the value function, as it has been empirically shown to perform well with visual inputs.

In DQN, the inputs are images and the output is  $Q(s, a)$ . We can treat this as a supervised learning problem, where now the target we aim to predict is  $y = r + \gamma \max_{a'} Q(s', a')$ , and we again aim to minimize the distance between this and our current estimate:

$$L = \|y - Q(s, a)\|^2. \quad (23)$$

Unlike traditional supervised learning methods, the target  $y$  changes because it is based on the current estimate since TD-learning bootstraps. One fix then is to use a target network for  $Q$  that is only updated every  $C$  steps. This allows the prediction to be more stable.

Another problem is that sequential samples are highly correlated, and are thus not i.i.d. The fix for this is to use an *experience replay buffer* to sample random experiences when making updates.

There is a trade-off for using deep learning to approximate value functions, as they are much slower to run and typically require specialized hardware such as GPUs. However, without this representation more basic approaches would likely fail in many scenarios.

### 2.3.6 Exploration vs exploitation

It is important in reinforcement learning for agents to gain enough experience from the world. An agent that aims only to maximize its immediate values may miss opportunities for higher reward regions. Reinforcement learning agents face a fundamental problem of whether to

exploit the information it already knows, or explore in order to gain new information.

In all of our approaches, we will use  $\epsilon$ -greedy for exploration. This approach takes random actions based on  $\epsilon$ , and  $\epsilon$  is either fixed to some low value or decayed over time. Other approaches like Boltzmann exploration have exploration built into the policies.

### 2.3.7 Reward specifications

The guiding signal behind learning desirable behaviors in RL is the reward. As such, it is an important and key component for properly solving RL problems.

One common problem in RL rewards are often only given at the end of a task. This is known as the *sparse reward* problem. For many tasks, it is impossible or very difficult to randomly reach the goal. Furthermore, it may take a long time for the algorithm to figure out which actions yielded the results. This problem is known as the *credit-assignment problem* and is one of the fundamental problems in reinforcement learning.

One approach to speeding up reinforcement learning is to introduce *shaping rewards*. However, if the reward is misspecified, then it can distract the agent and lead to locally-suboptimal policies. One way to remove the potential of such distractions is to use *potential-based shaping functions*.

## 2.4 Imitation Learning

Often, learning through rewards can be difficult, either because the reward is sparse and so it takes long to learn, or because we simply do not know how to define the rewards ourselves. In cases such as these, imitation learning is often used to remove the need to specify rewards at all. In this approach, experience is given in the form of demonstrations consisting of state-action tuples  $\langle s_1, a_1 \rangle \dots \langle s_n, a_n \rangle$ .

In the simplest form of imitation learning, given expert states and actions, we can use supervised learning to approximate  $\pi(a|s)$ . That is, given a state  $s$ , we aim to predict the probability of taking each action, i.e., the policy. This approach is known as *behavioral cloning*. To specify this as a supervised learning problem, we can represent the inputs as  $s$

and predict labels  $a$ .

## 2.5 Computer Vision

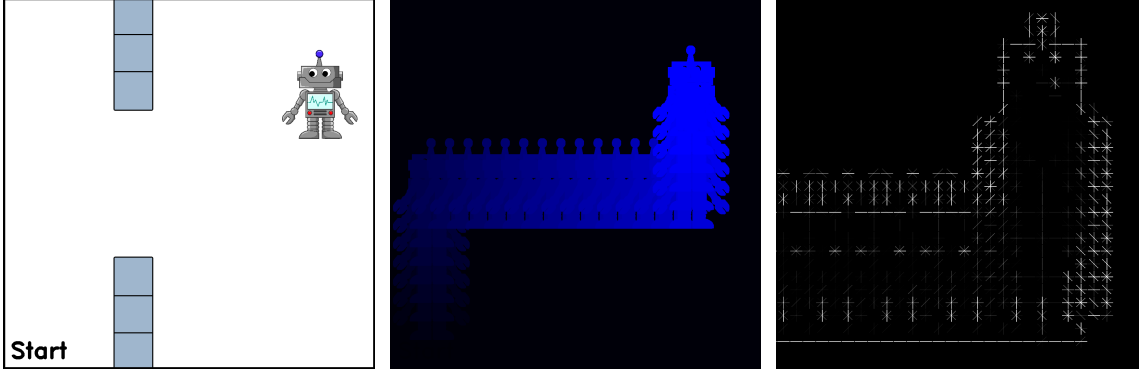


Figure 5: The image on the left shows the result of an agent moving from the **start** location to the top-right position. The middle image is the corresponding motion template. The image on the right is a zoomed-in visualization of the HOG features of the motion template.

Computer vision enables learning directly from images. Our work uses several computer vision approaches to develop and learn similarity metrics for reward functions and goal specifications. This section discusses the methods we will use in later chapters.

### 2.5.1 Motion templates

In Chapter 4, we will describe how to use motion templates to describe goals. A Motion Template (Figure 5) is a 2D visual representation of motion that has occurred in a sequence of images—typically from the segmented frames of a video [11, 22]. Movement that occurred more recently in time has a higher pixel intensity in the template than earlier motion and depicts both where and when motion occurred.

Calculating a motion template is an iterative process. The first step is to obtain a silhouette image of the motion that has occurred between each frame. The silhouette is computed by taking the absolute difference between two images and then computing the binary threshold, which sets all pixels below a threshold to 0 and all pixels above the threshold to 1.

A function  $\Psi(\mathbf{I})$  computes the motion template  $\mu$  for a sequence of images  $i_1, i_2, \dots, i_n \in \mathbf{I}$ . Let  $\sigma_t$  represent a silhouette image at time  $t$ . To calculate the motion template  $\mu$  of  $\mathbf{I}$ , we first compute a silhouette image  $\sigma_1, \sigma_2, \dots, \sigma_{n-1}$  between all consecutive images  $(i_1, i_2), (i_2, i_3), \dots, (i_{n-1}, i_n)$ . Then  $\forall_{x,y}$ , where  $x$  and  $y$  are respective column and row pixel locations, we can compute  $\mu_{t,x,y}$  for time  $t = 1, 2, \dots, n$ :

$$\mu_{t,x,y} = \begin{cases} \tau, & \text{if } \sigma_{t,x,y} > 0 \\ 0, & \text{else if } \mu_{t-1,x,y} < (\tau - \delta) \\ \mu_{t-1,x,y}, & \text{otherwise} \end{cases}$$

In words, the function increases the intensity of the pixel at  $x, y$  if movement has occurred at the current iteration  $t$ . Here,  $\delta$  and  $\tau$  are both parameters that influence how much  $\mu_t$  is decayed. The parameter  $\tau$  is a representation for the current time in the sequence and increases as  $t$  increases. The parameter  $\delta$  represents the duration of the motion template and controls how quickly pixels decay. Essentially,  $\Psi(\mathbf{I})$  layers the silhouette images and weights them by time.

### 2.5.2 HOG features

In chapter 4, we use hand-defined features to compute similarities between images. Often, taking distances over pixels yields poor results. For example, suppose two images are taken of the same object, but there is a shadow in one of them. The pixel values will be vastly different. Or perhaps one image is slightly shifted or scaled differently.

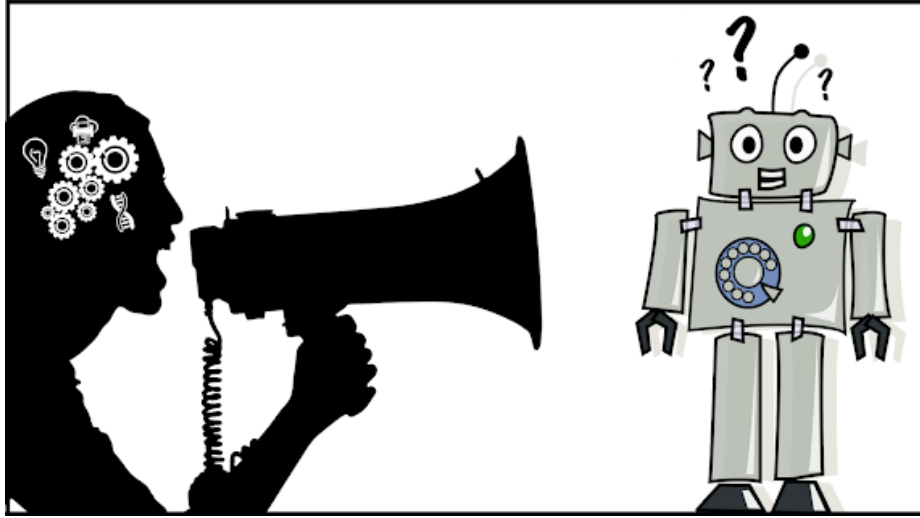
We often use feature descriptors to obtain a better representation of images. We use a Histogram of Oriented Gradients (HOG) [21], which is a feature descriptor that is capable of representing information about local appearances and shapes within images. The first step for computing HOG features is to calculate the gradients of the image, which account for changes in intensities between adjacent pixels. Then, the image is divided into cells, which help describe local information.

The next step is to compute a histogram of the gradients for each cell. Finally, the histograms are concatenated into a feature vector that represents the HOG features. A visual representation of the features can be found in Figure 5.

We will additionally use deep learning to learn features, which will provide a more general approach.

## Chapter III

### RELATED WORK



In this chapter, we will discuss many approaches related to those introduced in this dissertation. Because of the difficulties of defining hand-specified reward functions, we are interested in alternative mechanisms for expressing tasks to agents. This chapter will discuss several ways goal specifications have been used in reinforcement learning to elicit desired behaviors, and how imitation learning has been used both with and without actions to solve problems.

#### ***3.1 Goal specifications for reinforcement learning***

Reinforcement Learning (RL) agents traditionally rely on hand-designed scalar rewards to learn how to act. Unfortunately, the more complex and diverse environments and tasks become, the more difficult it may be to engineer rewards that yield desired behaviors from the agent. Experiment designers often have a goal in mind and then must reverse engineer a reward function that will likely lead to it. This process can be difficult, especially for non-experts. Furthermore, it is susceptible to reward hacking—unexpected and undesired

behavior that achieves high reward but does not capture the essence of what the engineer was trying to achieve [3]. Moreover, hand-designed reward functions may be brittle, as slight changes in the environment may yield large, and potentially unsafe, alterations in agent behavior.

The RL community has addressed these problems through many approaches including reward shaping [56], intrinsic rewards [15], hierarchical reinforcement learning [25, 79], and transfer learning [85]. Another approach however is to avoid designing scalar rewards altogether, and rather focus on specifying goals, for example, through imitation learning, target images, or multimodal channels such as speech and text.

We now describe several related methods for representing goals in RL. Throughout the literature, we have found that goals are often defined in terms of the agent’s environment. Our own approaches will introduce a general mechanism for specifying goals on our own terms. We discuss two broad techniques for representing goals, either by explicitly instantiating them through engineering or human feedback, or implicitly expressing them through demonstrations.

### ***3.2 Explicit goal instantiations***

Arguably, one of the most common approaches for representing goals for RL problems is to provide hand-specified rewards that are based on internal parameters of an agent’s environment. There is a wealth of literature that describes such domain-specific rewards, from those that indicate when an agent has reached a desired location [82], to more complex ones that can be used as feedback for training neural networks [98]. Rewards are largely where we instill domain knowledge, and so are inherently specialized. Even if a reward is commonly used for multiple problems in the same domain, it often remains a function of the configurations of each agent in the environment.

For problems consisting of visual inputs, a more general solution for representing goals is to use target images. With this approach, rewards are based on pixels alone, thus allowing



one to abstract away the task specification from the parameters of the agent’s configuration. Visual Servoing allows agents, particularly robots, to perform tasks that are represented through visual features [39]. While this approach is typically used for control, the idea of minimizing the error between a robot’s camera image and some goal image is similar to our own work. Another approach aims to minimize the difference between learned visual features from the agent’s state space and a target representation [30]. We are interested in examining how tasks can be represented even if the goal representation is not the same as the agent’s. In this case, learned features points about the agent’s state space might not map to those of the goal.

Images are a more natural representation for expressing tasks to agents than traditional methods. Many works have aimed to simplify goal representations. For example, there has been work in representing goals through hand-drawn sketches [73, 76]. Other works have focused on allowing goals to be represented through pre-existing sources. In one approach, a robot is trained to cook by showing it cooking videos [94]. Specifically, the robot learned the necessary grasps and actions to take through classification. Another work uses information from the web to train robots [86]. In this approach, robots could learn how to solve problems by parsing the language in instruction websites such as [wikihow.com](http://wikihow.com) and [ehow.com](http://ehow.com).

When multiple goals or MDP settings exist, it may be necessary to produce more general solutions. Early work aimed to find rewards that generalized across multiple environments [75]. Another work [32] described an approach that allowed generalizing skills learned from “labeled” MDPs with known reward functions to similar “unlabeled” MDPs with unspecified reward functions. Finally, we have seen approaches that aim learn a value function that generalizes over states and goals, but these approaches also assume that the two representations share a similar structure [68, 97].

Specifying the goal itself is not always difficult, but training an agent to actually solve the task may be. Training in simulation and then transferring the knowledge to the real-world can often be more efficient than training there directly (e.g. [63, 64, 88, 91, 95]).

Like the approaches in this dissertation, such specifications may also be considered cross-domain, although the motivation for these works is different from our own. In particular, such “sim-to-real” approaches tend to focus on transferring across separate realizations of similar domains. Hindsight Experience Replay [4] takes unsuccessful trajectories from an experience replay buffer and pretends that the end of the trajectory was actually the goal. This allows the agent to learn to reach the real goal more quickly. Recent model-based approaches use experiences imagined backwards from the goal to improve the experiences of the agent [27, 34].

Many methods utilize human feedback to guide an agent’s behavior. For example, some approaches allow humans to provide rewards to agents in real-time [42, 49, 87]. Alternatively, policy shaping takes a more hands-off approach, and provides policy “advice”, as opposed to explicit rewards [35]. Another approach aims to interactively teach an agent goals [47], or to allow the agent to learn based on which trajectories humans prefer [18]. Finally, rather than providing direct target images for where an object should be, another approach allows humans to select the pixel locations for where a robot should move an object [31].

Finally, there has been a large breadth of recent work that aims to specify goals in different modalities. Such approaches have similar a motivation to our own. That is, they aim to provide goal specifications in more natural settings than the agent’s. For example, sketches of maps have been used for representing desired trajectories in navigational tasks [12, 77], correspondences learned between words and robotic actions [80], rewards based on the touch of a handshake [61], and value functions learned from facial expressions [92]. Recently, an approach learned correspondences between written instructions and visual states in Atari games [44]. This approach is more similar to our own as it learns the correspondences between two different domain representations. Still, we have not yet seen an approach that learns a visual correspondence for cross-domain goal representations.

### 3.3 *Imitation Learning*

Imitation learning [67] is often used when specifying a goal is difficult, or when a problem is too challenging for an agent to solve on its own. It is often difficult to determine which states to assign credit to in reinforcement learning, especially when the reward is sparse. Imitation learning addresses this by allowing experts and non-experts alike to demonstrate how to solve the problem. These trajectories can provide a huge boost in training time for agents.

Imitation learning is the field of training artificial and real-world agents to imitate expert behavior using a set of demonstrations. This approach has an extensive breadth of applications and a long history of research, ranging from early successes in autonomous driving [60], to applications in robotics [16, 66] and software agents (e.g. [55, 74]). However, these approaches typically assume that the expert’s actions are known. This often requires the data to be specifically recorded for the purpose of imitation learning and drastically reduces the amount of data that is readily available. Furthermore, approaches that do not require expert actions typically must *first* learn behaviors in the agent’s environment through extensive interactions. As we will discuss, we aim to learn to imitate from only observations of expert states, followed by only a few necessary interactions with the environment. We now describe classic approaches to imitation learning along with more modern approaches.

#### 3.3.1 **Classic approaches**

Arguably, the most straight-forward approach to imitation learning is behavioral cloning [60], which treats imitation learning as a supervised learning problem. More sophisticated methods achieve better performance by reasoning about the state-transitions explicitly, but often require extensive information about the effects of the agent’s actions on the environment. This information can come either in the form of a full, often unknown, dynamics model, or through numerous interactions with the environment. Inverse Reinforcement Learning (IRL) achieves this by using the demonstrated state-action pairs to explicitly derive the expert’s

intent in the form of a reward function [1, 57].

### **3.3.2 Correspondence problems**

Imitation learning methods often assume that the student, i.e., agent, and the teacher, or demonstrator, share a common environment. That is, the observed states and actions are shared between the teacher and student. When the observations are dissimilar, a correspondence problem must be solved [5]. These correspondences are often hand-specified, and may require equipment such as sensors on the teacher’s body. Our own approach aims to learn such correspondences between different representations.

Recent works have proposed solving the correspondence problem automatically. For example, one approach used deep learning to train an agent that had a first-person perspective through third-person samples [78]. Another somewhat related RL approach is transfer learning, which aims to transfer learned behaviors from one domain to another [85], for example by initializing the parameters of policies in unsolved tasks [2], or by transferring skills across untrained robots [23]. Time-contrastive networks [71] finds a correspondence between videos of humans and robots by assuming that the sequences are aligned by time.

### **3.3.3 Direct policy optimization methods**

Recently, more direct approaches have been introduced that aim to match the state-action visitation frequencies observed by the agent to those seen in demonstrations. GAIL [38] learns to imitate policies from demonstrations and uses adversarial training to distinguish if a state-action pair comes from following the agent or expert’s policy while simultaneously minimizing the difference between the two. SAIL [69] achieves a similar goal by using temporal difference learning to estimate the gradient of the normalized state-action visitation frequency directly. However, while these approaches are efficient in the amount of expert data necessary for training, they typically require a substantial amount of interactions within the environment.

### 3.3.4 Learning from state observations

Increasingly, works have aspired to learn from observation alone without utilizing expert actions. Imitation from Observation [51], for example, learns to imitate from videos without actions and translates from one context to another. However, this approach requires using learned features to compute rewards for reinforcement learning, which will thus require many environment samples to learn a policy. Similarly, time contrastive networks [71] and unsupervised perceptual rewards [72] train robots to imitate from demonstrations of humans performing tasks. But this also learns a reward signal that is later used for reinforcement learning. Therefore, while these approaches learn policies from state observations, they require an intermediary step of using a reward signal, whereas we learn the policy directly without performing reinforcement learning.

Recent works have aimed to learn from observations by first learning how to imitate in a self-supervised manner, then given a task, attempt it zero-shot [59]. However, this approach again requires learning in the agent’s environment rather than initially learning from the observations. Another approach utilizes learned inverse dynamics to train agents from observation [89]. However, while this approach aims to minimize the number of necessary interactions with the environment after a demonstration has been provided, it only shifts the burden to a preprocessing step as learning the inverse dynamics model usually still requires a substantial number of interactions with the environment. Our work aims to first learn policies from demonstrations *offline*, and then only use a few interactions with the environment to learn the true action labels. Finally, recent work used audio to align different Youtube videos to train an agent to learn Montezuma’s revenge and Pitfall [6].

### 3.3.5 Multi-modal predictions

In chapter 7, we will describe a method that predicts next states given a state and latent action. This is similar to recent works that have learned action-conditional predictions for reinforcement learning environments [17, 58], but those approaches utilize ground truth

action labels. Rather, our approach learns a latent multi-modal distribution over future predictions. Other related works have utilized latent information to make multi-modal predictions. For example, BicycleGAN [96] learns to predict a distribution over image-to-image translations, where the modes are sampled given a latent vector. InfoGAN uses latent codes for learning interpretable representations [14], and InfoGAIL [50] uses that approach to capture latent factors of variation between different demonstrations. These works, however, do not attempt to learn direct priors over the modes, which is crucial in our formulation for deriving policies.

As such, our approach is more analogous to online clustering, as we aim to learn multiple expected next states and priors over them. However, we do not have direct access to the clusters or means. Other works have aimed to cluster demonstrations, but these approaches have traditionally segmented different types of trajectories, which represent distinct preferences, rather than next-state predictions [7, 37].

## Chapter IV

### PERCEPTUAL REWARD FUNCTIONS

In this chapter, we will begin discussing the contributions of this dissertation in more detail. First, we will examine how to use single images to specify goals. Representing goals in this way will put less constraints on the agent, and will require less engineering knowledge and demonstration effort of the demonstrator.

The purpose this chapter is not to demonstrate that goals can be represented through images. As we just discussed in the previous chapter, there are several approaches for this type of representation. Rather, we aim to show that we have control over where these goals are obtained from and how they are represented. In this chapter, we demonstrate how goals can be represented through motion obtained from humans. By defining a domain-agnostic correspondence, we show that this goal specification can be used to train a simulated robot.

The next two chapters address problems that we refer to as **emulation learning**. In emulation learning, we are given a single image of a goal. Unlike imitation learning, we do not aim to match an expert’s behavior, only the end-goal. We focus on emulating from goals obtained from outside of the agent’s environment.

#### ***4.1 Introduction***

Raw pixels have become a popular state representation for reinforcement learning problems [54]. Previously, state design required hand-defining domain-specific components such as the exact position information of the agent and other objects. For example, to represent the state for the game Mario, we would need to encode the location of the player, the blocks in the level, and each of the enemies. Using images for state representations allows us to abstract away the design of the relevant features of the agent’s task.

Nevertheless, rewards have often still been defined through parameters that are internal

to the domain implementation. This representation is very specialized, and often needs to be re-specified each time the task changes.

We are interested in a more general reward function that is based only on visual features. There are many advantages to using this type of representation:

1. It does not require manipulating domain-specific parameters when the task changes;
2. It is more representative of how humans specify problems;
3. We can define similarities that allow specifying goals that are agnostic to the agent’s environment.

In this chapter, we introduce Perceptual Reward Functions (PRFs), where the reward is based on how similar an agent’s visual representation—such as an image from a camera or simulation—is to some goal representation. By hand-defining a single similarity metric for a task, we show that we only need to modify a perceptual task specification when the goal changes, while the reward function remains fixed. Furthermore, it allows us to compute functions that are agnostic to the agent’s environment, thus removing the constraint that the goal must be represented in the agent’s environment. We will describe two approaches for representing tasks in this manner. Our experiments will focus on training a simulated robot to mimic human facial expressions.

## **4.2 Approach**

In this chapter, we aim to provide a mechanism for describing goals across domains without engineering domain-specific scalar rewards. When we express tasks, we often must consider the configuration of the agent and its environment. For example, to define a reward for completing a level in Mario, we would likely need to have access to the game simulator or use domain-specific computer vision techniques to extract the score.

Rather than declaring tasks as being complete when specific configurations are met, perceptual task descriptions allow one to represent how the completed task should *look*.



Such an approach would be useful if the task requirements were difficult to program or if we did not have access to the agent’s internal configuration, for example if we were an end-user teaching an agent to learn tasks in multiple environments. We now define some terminology for our approach and then describe methods for visually describing tasks.

### 4.2.1 Formalities

We aim to solve problems represented through an MDP, where the states consist of images, the environmental reward function is unknown, and the transition function is unknown. We aim to use PRFs to replace the environmental reward.

We refer to a *Task* as a member of a class of problems that need to be solved, and a *Goal* as an instantiation of a task. For example, building origami figures may be the task, whereas constructing origami frogs, cranes, etc. are the goals.

We define a *Perceptual Template* as an image that is used to represent an agent’s state or goal. We call the agent’s environmental state, represented through a perceptual template, its *Mirror State*  $T_M$ . We define a *Goal Template*,  $T_G$ , as a perceptual template of the agent’s goal. We define an *Agent Template*,  $T_A$ , as a perceptual template derived  $T_M$  that will be compared to  $T_G$  to obtain a reward.

We later describe two different representations for perceptual templates (Section 4.2.3). In the next section, we describe how to compute rewards given these templates.

In summary, the assumptions made in this chapter are:

#### Assumptions:

- ◇ States consist of images;
- ◇ The environment reward function is unknown;
- ◇ The transition function is unknown;
- ◇ Single goal specification.

### 4.2.2 Perceptual reward functions

In traditional reinforcement learning scenarios, the reward function is engineered for each task and must be changed for each new problem. We are interested in general reward functions that are not based on domain-specific engineering. A reward function based on images represents a general reward function, as only the inputs, not the function, changes across goals and tasks. In particular, we are interested in developing Perceptual Reward Functions (PRFs).

**Definition 4.2.1** *Perceptual Reward Function* A function that computes a reward that represents how similar  $T_A$  is to  $T_G$ .

There are many ways we can represent a PRF, and in the next chapter we demonstrate how it can be learned. The benefit of this reward function is that it is now a function with images as inputs, and so it is not dependent on internal domain variables. In this chapter, we focus on hand-defining a PRF:

$$F(T_A, T_G) = \frac{1}{e^{D(T_A, T_G)}}, \quad (24)$$

where  $F$  represents the perceptual reward function and  $D$  is a distance metric. We use an exponential function to avoid dividing by 0 and to separate rewards with similar distances. This reward function ensures that as the distance between  $T_A$  and  $T_G$  decreases, the reward will become larger. This representation of the reward is dense, but could be made sparse by only giving at terminal states.

The smallest distance that can be returned by  $D$  is 0, and so  $F$  will return rewards that are between 0 and 1. As the distance between  $T_A$  and  $T_G$  increases, the output of  $F$  will approach 0. An optimal policy then should return actions that minimize the visual distance between  $T_A$  and  $T_G$ . We now describe how  $D$  can be computed.

Taking distances over pixels typically yields poor performance. In our case, because we are interested in cross-domain representations, this will be especially true. As such, we use

a function  $H(T)$  to compute HOG features, which were described in chapter 2, of images before computing the distance. One reason to use this feature descriptor is that the approach divides the image into cells, thus accounting for some differences in translation and scale. Additionally, the descriptor has no information about specific image features, such as color or texture. Essentially, we can use HOG features to compare  $T_A$  and  $T_G$ , even if they come from different sources.

For some tasks, we might wish to increase the number of cells used with HOG features for more acuity. However, increasing the number of cells also increases the time to compute the features. Thus, we keep the cell size as a parameter that can be input into the PRF. A simple approach is to set the cell size to be some fraction of the height,  $h$ , of the cropped region computed by  $H$ .

We now define the distance metric:

$$D(T_A, T_G) = \|H(T_A) - H(T_G)\|^2 \quad (25)$$

In words, we take the Euclidean distance between the HOG features of the cropped templates. In the next section, we discuss how one can obtain  $T_A$  and  $T_G$ .

### 4.2.3 Perceptual task descriptors

One benefit of our approach is that it allows one to focus on task representation, rather than reward engineering. In order to create a PRF, one simply needs to define  $T_A$  and  $T_G$ . We now outline two different task representations.

#### 4.2.3.1 Mirror task descriptors

The first task representation we consider requires that  $T_A$  be based on the agent's mirror state  $T_M$ . We call this type of representation a *Mirror Task Descriptor*. In the simplest form of a mirror task descriptor,  $T_A = T_M$ . We call such a representation a *Direct Task Descriptor*, as we can compare  $T_G$  directly with the mirror state. This representation is convenient if

we can represent how the agent’s entire mirror state should look at the end of a task. As discussed in Chapter 3, this type of representation has been used before to represent goals, but approaches typically assume that  $T_A$  and  $T_G$  are drawn from the same distribution or have some similar features. We will examine how well this approach works when the two environments are significantly different.

#### 4.2.3.2 *Motion template task descriptor*

We also consider tasks that are based on a trajectory of motion. A *Motion Template Task Descriptor* represents tasks using motion templates. As we noted in Chapter 2, a motion template is constructed from a sequence of images. We will soon describe how we can obtain two image sequences,  $\mathbf{G}$  and  $\mathbf{S}$ , that will be used to compute  $T_G$  and  $T_A$ , respectively.

Using motion templates for PRFs is appropriate when the goal can be represented through a trajectory of motion. One benefit of using a motion template for a PRF is that it does not require knowledge of domain-specific features, thus allowing for task generality. In fact, motion templates contain *no* information about specific features, such as texture or color, thus allowing the source image sequence for  $\mathbf{G}$  to be different than that of  $\mathbf{S}$ . For example, a person could create a sequence of images by recording herself performing the task, or a pre-existing image sequence, such as frames from a video, could be used. Depending on the agent representation, one might even be able to use a video of an animal or a cartoon character. Furthermore, because we are using HOG features, we can obtain information about where movement should occur and so the agent should be motivated to take actions in the correct regions of its environment.

Now, we describe the simple process of obtaining the agent’s image sequence  $\mathbf{S}$ . At the beginning of each episode, we initialize the sequence such that  $\mathbf{S} = \{\}$ . Each time the agent takes a step in an episode, its mirror state  $T_M$  is added to  $\mathbf{S}$ .

Given the goal and state sequences, we can compute their respective motion templates. That is,

$$T_G = \Psi(\mathbf{G}), \quad (26)$$

$$T_A = \Psi(\mathbf{S}). \quad (27)$$

### 4.3 Experiments and results

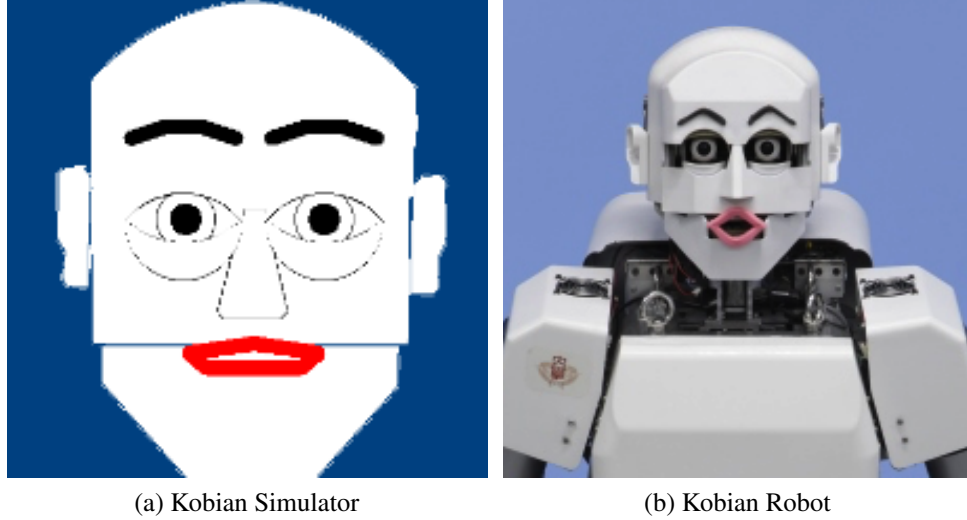


Figure 6: The Kobian environment. In the Kobian Simulator, the agent must move parts of its face to make expressions.

Our experiments aimed to show that PRFs are a general and feasible task representation for reinforcement learning. In particular, we aim to show that an agent is capable of learning from a visual task descriptor that did not require engineering scalar rewards that were based on domain-specific parameters.

#### 4.3.1 Kobian Environment

In our experiments, we aimed to train an agent to learn how to make facial expressions. Such a skill could be used for animating virtual characters or for real robots that display emotions. The environment used in the experiments is a simulation of the Kobian-RII robot’s face, shown in figure 6 [48]. Kobian’s head has 24 degrees of freedom, allowing the robot to move its eyebrows, eyes, and mouth either up, down, left, or right. The agent’s state variables

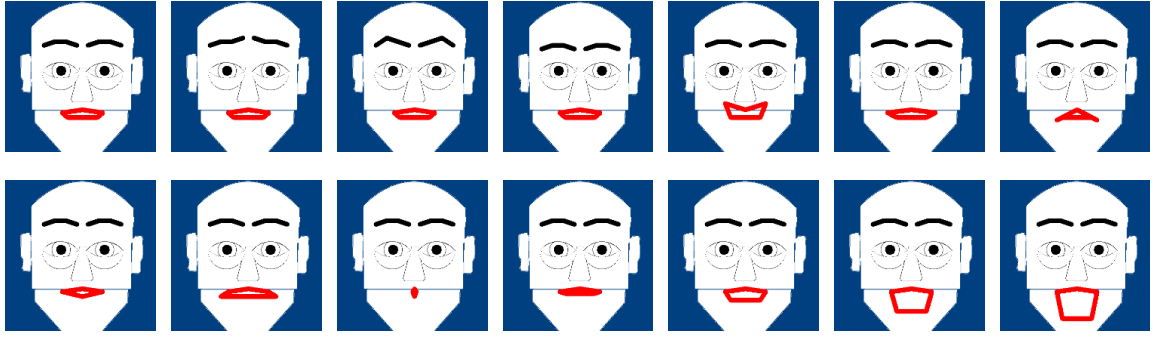


Figure 7: Action Units (AUs) applied to Kobian’s face. The top left shows Kobian’s neutral face. The remaining top row images represent actions: Inner Brow Raiser, Outer Brow Raiser, Brow Lowerer, Upper Lid Raiser, Upper Lip Raiser and Lip Corner Puller. The bottom row represents actions: Lip Corner Depressor, Chin Raiser, Lip Stretcher, Lip Tightener, Lip Pressor, Lips Part, Jaw Drop, Mouth Stretch. Additionally, the agent can take the No-Op action  $N$ . Note that these are the actions applied to Kobian’s neutral face. Each action can be applied on top of another to obtain faces not shown here.

represent values for each of these components of the face.

Facial expressions are often described through Action Units (AUs) that characterize the motion that created the expression. To reduce the action space, we model the agent’s actions after these AUs. This is feasible, as it has been found that people can recognize AUs applied to a robot’s face [65, 90]. The motions the agent has available are described in figure 7.

Kobian’s face is initialized to the neutral face shown on the right in figure 6. When Kobian takes an action in the simulator its state variables are changed. The AUs can be applied multiple times until the bounds on the state variables are reached. We take a combination of AUs on the simulation that are commonly used for describing the facial expression on humans. We limited the agent’s episode to 10 steps and allowed it to take a no-op action that did not affect its state.

Because there are many facial expressions, this would be a difficult skill to engineer rewards for or to demonstrate to the agent. Thus, we used two PRFs for this task. For the first PRF, which we call the motion PRF, we used motion templates computed from videos from the Cohn-Kanade database [43, 52], which consists of videos of humans making facial expressions. Here,  $T_A$  is the agent’s motion template and  $T_G$  is the human’s. For the second

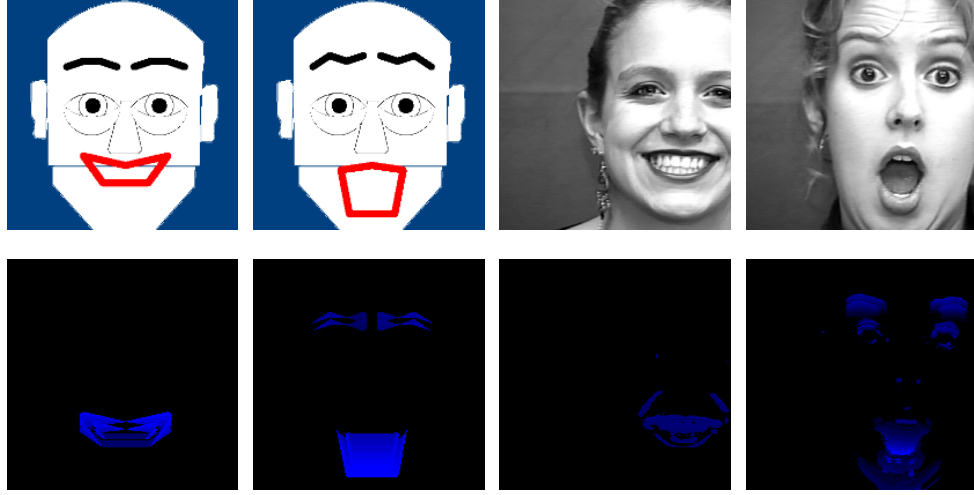


Figure 8: Goal representations for Kobian. The faces of Kobian are the desired goals for the standard reward function, which is based on the internal variables of the agent. The human faces (©Jeffrey Cohn) are the desired goal for the face PRF, and the motion templates below are the desired goals for the motion PRF.

PRF, which we call the face PRF, we used the direct face of the human, which was extracted from the last frame of the video. Here,  $T_A$  is the robot’s face and  $T_G$  is the human’s. We show these goal templates in figure 8. Note that these faces are far from the ideal case—the images are off-center, the lighting conditions are different, the participants are wearing jewelry and have hair, which is much different from the kobian representation.

We compare our work against a traditional reward setup where rewards are a function of the agent’s state variables. The reward becomes the inverse Euclidean distance between the agent’s state variables and the goal variables. We call this reward function the standard reward in our experiments.

We used DQN to train the agents to make happiness and surprise faces. More details of the experiments can be found in the appendix.

#### 4.3.2 Results

In our experiments, we aimed to answer the following questions:

1. *Do PRFs enable reward maximization?*

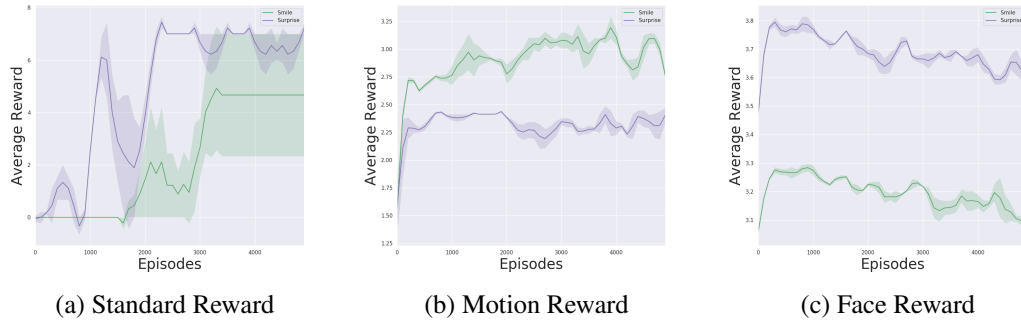


Figure 9: The rewards obtained in Kobian for each type of reward function. We ran the Kobian simulator for 2500 episodes for each experiment and averaged over 3 runs. We plot each graph separately because the rewards occur on different scales.

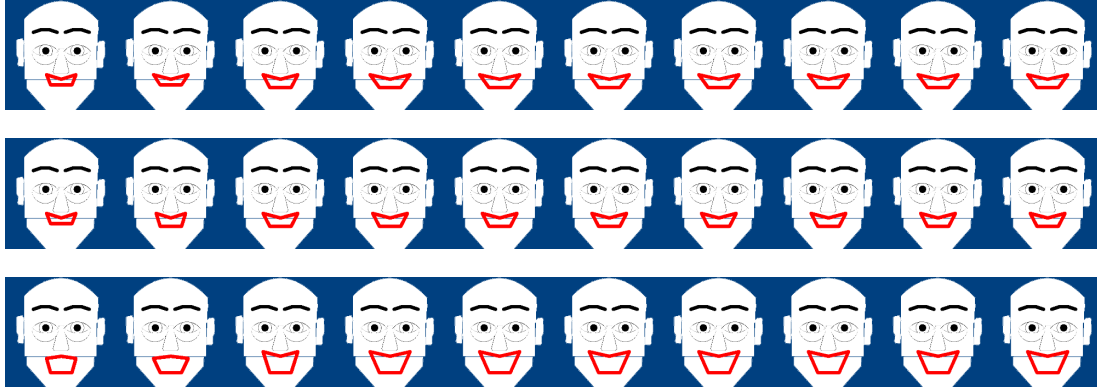


Figure 10: Sequence of highest rewarded “happiness” faces for standard (top), motion (middle) and face (bottom) reward functions.

## 2. Do the approaches learn the correct policy?

**Do PRFs enable reward maximization?** We first aimed to examine whether each reward function enabled reward maximization for reinforcement learning. This is not a common question in RL problems, we wanted to determine if the rewards returned were not some constant number and that the agent moved towards *some* goal. These results are shown in figure 9. It is clear that the rewards do in fact increase over time. Because each approach used a different metric for its reward, we do not compare those values directly.

**Do the approaches learn the correct policy?** We show the faces each agent learned for happiness in figure 10 and for surprise 11. We measure correctness subjectively by how each face looks, but also quantitatively based on how close the internal variables of each





Figure 11: Sequence of highest rewarded “surprise” faces for standard (top), motion (middle) and face (bottom) reward functions.

approach match the standard approach, which was based on how we define human action units.

The happiness emotion consists of pulling the corners of one’s lips. We can see that each representation learned to do these actions. The face learned by the motion PRF almost completely matched that of the standard goal. Furthermore, we see in figure 12 how closely each approach matches the internal variables of the desired goal. It is clear that the smile motion template was a good representation as the distance decreases as it learns.

The face PRF yields a very different outcome for the happiness emotion, as the mouth is dropped quite a bit more. This is interesting because the motion and face PRF are based on the same face. Furthermore, the distance to the desired goal does not decrease. In this case, using motion as the representation yielded a closer face to the desired action units.

The surprise emotion consists of raising one’s eyebrows and dropping their mouth. The standard reward function yielded the desired goal, but both PRF representations learned undesirable goals. While the motion PRF did learn to raise its eyebrows and drop its mouth, it eventually closed it. This is an unfortunate consequence of using motion as the representation, as it is difficult to tell in the motion template when sequences overlap. One way to address this would be to make the template decay more quickly.

The face PRF did not learn to raise its eyebrows. In fact, this representation yielded a very similar face to that of the smile task, which may indicate that this representation is not

sufficient for the task of making facial expressions.

Overall, we have shown that motion template descriptors can be a good representation for PRFs, even when the source of  $T_A$  is different than that of  $T_G$ . However, some tuning is necessary to enable better learning.

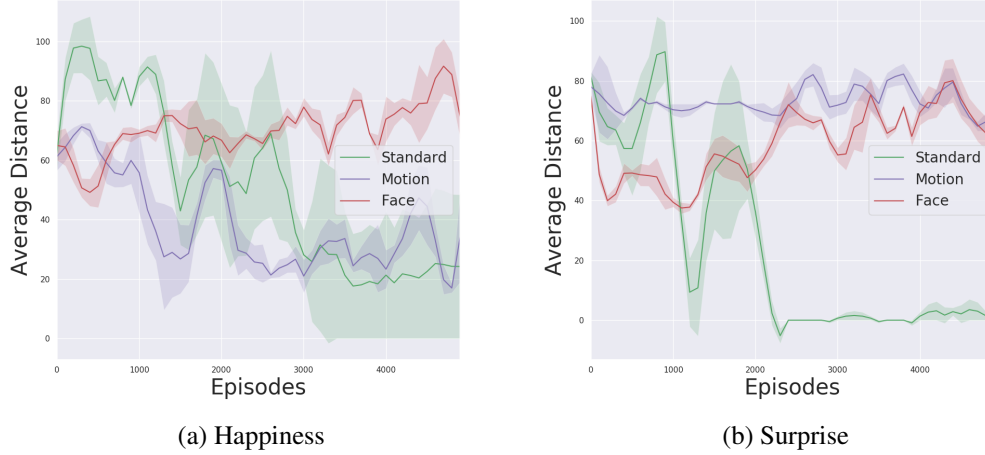


Figure 12: The distance  $D$  between the HPRF goal template,  $T_G$ , and the motion templates computed during each episode of the VRF, KPRF, and HPRF happiness and surprise tasks.

#### 4.4 Discussion and conclusion

In this chapter, we have shown that PRFs allow one to create visual task descriptions without modifying domain-specific reward parameters. We introduced a motion task representation and have shown empirically that they can yield desirable behavior. We have demonstrated that the reward function is general and still allows tasks to be solved. We demonstrated how to train an agent from an alternative environment without requiring engineering a reward based on internal task parameters.

To summarize the contributions introduced in this chapter, we have demonstrated that:

1. We can represent goals through images that we choose that are agnostic to the agent's state representation, internal reward, and action space, thus demonstrating that this is as straightforward domain-specific rewards;

2. PRFS only needed to be specified once, thus demonstrating generality;
3. The agent could perform as well as the standard agent, thus demonstrating that PRFs representation equally enables task completion.

One limitation is that the representation had to be hand-specified and needed to be tuned. The focus of the rest of the work will be to learn reward functions and policies, and in the next chapter, we will describe how to learn correspondences from cross-domain representations.

## Chapter V

### CROSS-DOMAIN PERCEPTUAL REWARD FUNCTIONS

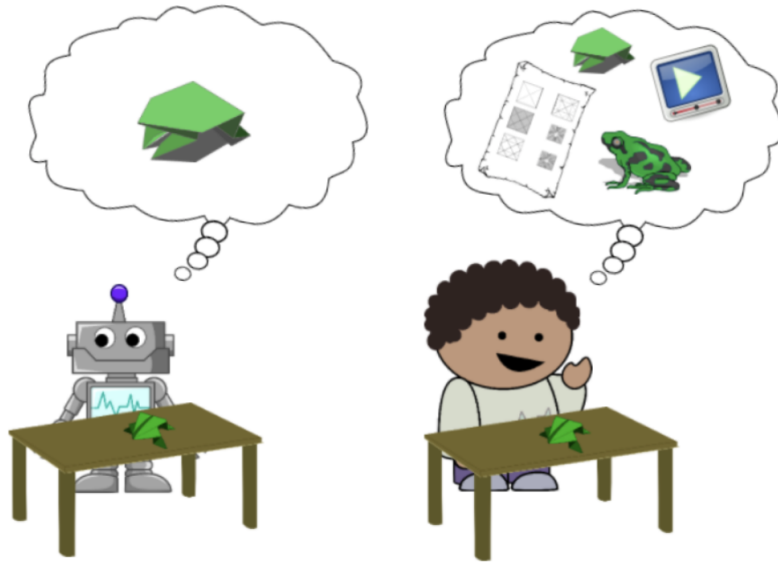


Figure 13: An example of how a reinforcement learning agent (left) typically learns tasks, versus a human (right). Agents typically have goals that are specified in its own configuration space, whereas humans can learn from many different task representations.

This chapter continues our discussions about utilizing single images for representing perceptual goals for emulation learning. In the previous chapter, we described how we could represent tasks with a single example. However, it required manually defining a reward function and representations that allowed the agent and goal templates to be compared.

To enable learning from many situations, we should allow expressing goals from outside representations without requiring manually defining a common representation. As such, in this chapter, we describe how we can *learn* a correspondence and a corresponding perceptual reward function.

## 5.1 Introduction

A common technique humans use when defining problems is to utilize visual replicas of the solved instance. We may, for example, describe how to build a table by taking an picture of one that is already configured. This is analogous in RL terms to specifying a reward based on the agent’s environment, either through the parameters of the agent’s state or target images. One problem with this approach is that it requires some understanding of the solution in the goal environment. For people, it can often be easier to describe solutions through other mediums, for example through written instructions, verbal speech, diagrams, etc. This allows us to define the goal on our own terms without needing to know the solution in a manner that may be difficult to specify. For example, we might not physically be capable of demonstrating how to build a table, but we may be able to tell someone verbally how to construct it.

We aim to allow specifying tasks in similar manners—across domains—through cross-domain goal instantiations that are defined in environments outside of the agent’s. Rather than hand-specifying rewards based on internal parameters of the agent, or providing target images from the agent’s environment, our approach allows specifying goals in surrogate environments, where one may more easily find or construct solutions. This allows one to specify a goal without knowing anything about the internal configuration of the agent. Unlike in the previous chapter, we are interested in *learning* a similarity through supervised learning, rather than hand-defining it. The advantage of this approach over traditional methods is that:

1. It allows specifying goals in surrogate environments where solutions can more easily found or constructed;
2. It learns a correspondence between environments;
3. It learns a reward function rather than hand-specifying it.

In this chapter, we introduce Cross-Domain Perceptual Reward (CDPR) functions, which produce rewards that represent deeply learned similarities between a cross-domain goal image and states from an agent’s environment. We empirically demonstrate that CDPRs are a *general* approach for providing *adequate* rewards between these cross-domain and intra-domain representations.

We provide evidence of generality by showing that CDPRs can be used without modification across multiple goal instantiations. We demonstrate that CDPRs can successfully train an agent to complete a task. We demonstrate this approach within a maze environment with two distinct cross-domain goal specifications. We show that by using CDPRs as a replacement for hand-specified reward functions, we can successfully train an agent to solve these tasks with deep reinforcement learning, with the added benefit that we can specify the goals on our own terms.

## 5.2 *Approach*

We now formalize our approach for developing Cross-Domain Perceptual Reward (CDPR) functions.

## 5.3 *Formalities*

We are interested in specifying goals through images obtained from alternative, cross-domain environments. Unlike in the previous chapter, we now aim to *learn* the correspondence between the agent and goal templates by using examples from an expert.

We aim to solve problems where an agent has a task with multiple potential goal configurations. An agent’s overall task may be to build furniture, for example, but its goal could be to build a chair, or table, or bench. Put succinctly, each goal is a specific instance of a task. Typically, an agent’s task is defined by a reward function. This remains true for our approach as well, but now we require the reward to be a function of both the state and a goal. Therefore, for a single task, this *general* reward function remains fixed when the goal

changes—only the inputs vary.

As in the previous chapter, we represent problems through an MDP, where the states consist of images and the environment reward function and transition function are unknown. Now, we aim to use CDPRs as a replacement for the reward.

In summary, the assumptions made in this chapter are:

**Assumptions:**

- ◇ States consist of images;
- ◇ The environment reward function is unknown;
- ◇ The transition function is unknown;
- ◇ Single cross-domain goal specification;
- ◇ Expert examples of goal pairs.

## ***5.4 Cross-domain perceptual reward functions***

In this section, we describe how we construct CDPRs.

### **5.4.1 Network representation**

We will make use of an approach that uses deep learning to find cross-domain similarities between images and their corresponding spoken captions, which are represented through spectrograms [36]. We hypothesize that this approach can be used in RL to represent cross-domain goals. The joint audio-visual approach used a network tailored to the spoken captions, but we aim to use a more general network that can accept any image as input.

A diagram of the network is shown in Figure 14. The inputs to the network are states from the agent’s environment,  $G_i$ , and cross-domain goal images,  $\hat{G}_j$ . We use this notation

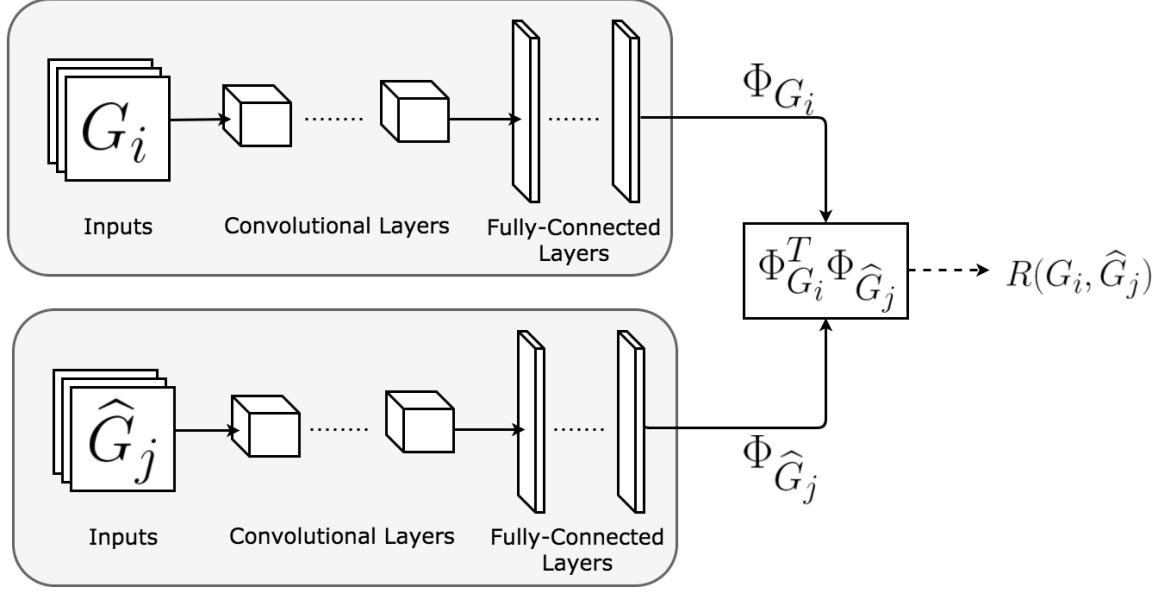


Figure 14: Deep neural network used for learning Cross-Domain Perceptual Reward (CDPR) functions. Two separate, though identical, networks encode features for the agent states  $G_i$  and cross-domain goals  $\hat{G}_j$ , respectively. Each network consists of a set of convolutional layers followed by a set of fully-connected layers. The exact components of each layer are dependent on the problem. The dot product between the outputs of each encoding layer represents the CDPR,  $R(G_i, \hat{G}_j)$ .

for the agent’s state from now on, as we consider each state image to be a potential goal image. The role of the CDPR is to determine this by outputting similarities between the two.

We construct two separate networks with identical architectures to encode the intra-domain features,  $\Phi_{G_i}$ , and the cross-domain features,  $\Phi_{\hat{G}_j}$ , respectively. The CDPR is obtained by taking the dot product of these two outputs:

$$R(G_i, \hat{G}_j) = \Phi_{G_i}^T \Phi_{\hat{G}_j} \quad (28)$$

This reward then acts as the similarity between an agent’s state and a cross-domain goal.



### 5.4.2 Training mechanism

Assume that for a given task, that we have  $k$  pairs consisting of an intra-domain goal specification and a corresponding cross-domain goal:  $\{(G_1, \hat{G}_1), \dots, (G_k, \hat{G}_k)\}$ . After training, we only use cross-domain images to instantiate goals. We aim to learn a reward function that makes  $R(G_i, \hat{G}_j)$  large when  $i = j$  and small otherwise. As such, we use the loss function from the previously described approach, which optimizes for exactly this target, modified appropriately for learning reward functions. Given the parameters  $\theta$  of the network described in Figure 14, the loss can be formulated as:

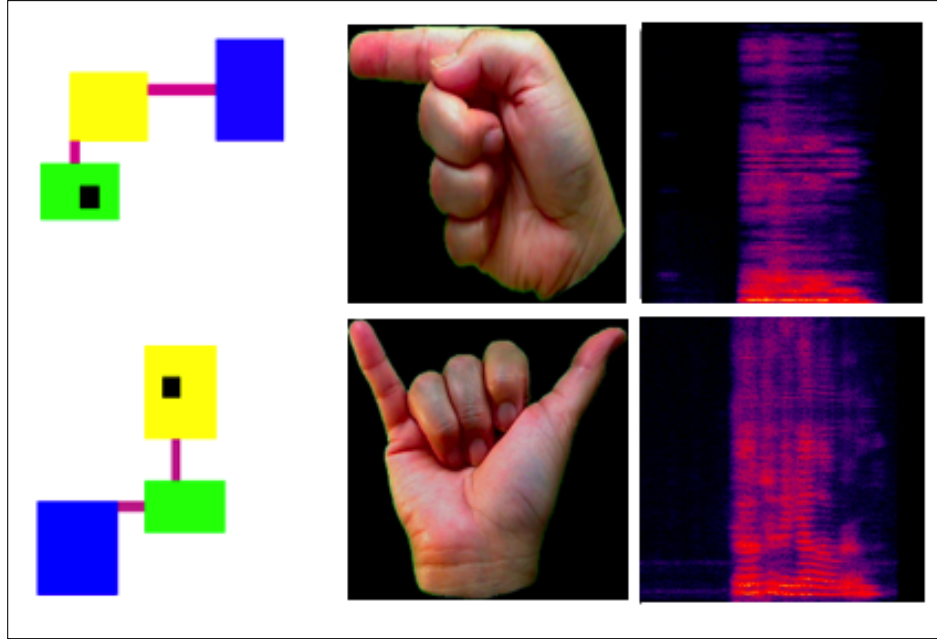
$$L(\theta) = \max(0, R_\theta(G_j, \hat{G}_i) - R_\theta(G_i, \hat{G}_i) + 1) + \max(0, R_\theta(G_i, \hat{G}_j) - R_\theta(G_i, \hat{G}_i) + 1) \quad (29)$$

We use gradient descent to optimize this loss function. For each training batch, we randomly sample two goal pairs,  $(G_i, \hat{G}_i)$  and  $(G_j, \hat{G}_j)$ . The loss aims to make the reward for the matching pair,  $R(G_i, \hat{G}_i)$ , be larger than the rewards for mismatched pairs,  $R(G_j, \hat{G}_i)$  and  $R(G_i, \hat{G}_j)$ . Otherwise, the outputs for the loss function will be greater than 0 and so the gradients will be penalized. Once we obtain the CDPRs, we can use them to instantiate the reward function for the MDP, and use standard RL approaches to solve the task. We should note that while the CDPR remains fixed across goal specifications, it is necessary to learn a new CDPR for each type of cross-domain goal representation.

## 5.5 Experiments and results

We aim to demonstrate the generality of CDPRs. We compare against a standard hand-specified reward, and Imitation from Observation (IFO) [51], which learns to translate from a source to target domain, and represents the reward as the negative distance between extracted features. For our experiments, we automatically generated the cross-domain goal representations.

We aim to measure how well the CDPRs work with RL. To evaluate generality, we used



(a) Maze Domain

Figure 15: Environment used in RL experiments. In the Maze domain, the agent’s task is to navigate to a specific room. The goal specifies the room the agent needs to navigate to. For the cross-domain goals, we used a sign language handshape of the first letter of the desired room’s color, and a spectrogram of a spoken command, “Go to the [color] room.” . The leftmost column represents the intra-domain goal representations, while the remaining two are the cross-domain goal representations. The top and bottom rows are referred to as Goal 1 and Goal 2 in our experiments.

two unseen goal instantiations for each task, as shown in Figure 15. We used Deep RL to solve the tasks with the architecture described in the paper [53]. We trained the network with an Adam optimizer with an initial learning rate of  $10^{-4}$  and a batch size of 32. To evaluate the agent’s performance, we ran its learned policy on the task every 100 episodes and measure the average reward obtained.

### 5.5.1 Maze environment

We evaluate our approach in a maze environment, as shown in Fig 15a. We use procedurally generated mazes consisting of 1-3 green, blue, or yellow colored rooms. The agent’s actions are to move up, down, left or right, and its task is to navigate the maze. Goals designate the specific room the agent needs to reach. In order to specify the desired room using standard RL approaches, we would need to know its coordinates. However, in the general case, it is unlikely that we would know the exact location of each desired goal. As such, we utilize two cross-domain goal representations: a sign language handshape indicating the first letter of the desired room color, and a spectrogram of a spoken command stating “Go to the [color] room.”

We use a dataset of sign language gestures from [8] for the handshape specification. For the speech specification, we recorded one sample of the spoken command for each colored room, and then varied the pitch to produce multiple samples. We set the reward for the standard approach as 1 if the agent is located in the desired room and 0 otherwise. The agent’s episode resets after 50 steps. During training, the agent could be initialized in any location on the map. During evaluation, it is initialized in the green room in the top maze in Figure 15a and in the yellow room in the bottom maze. More experiment details can be found in the appendix.

### 5.5.2 Results

In this section, we discuss results for training the CDPRs for each task and running RL with the learned rewards.

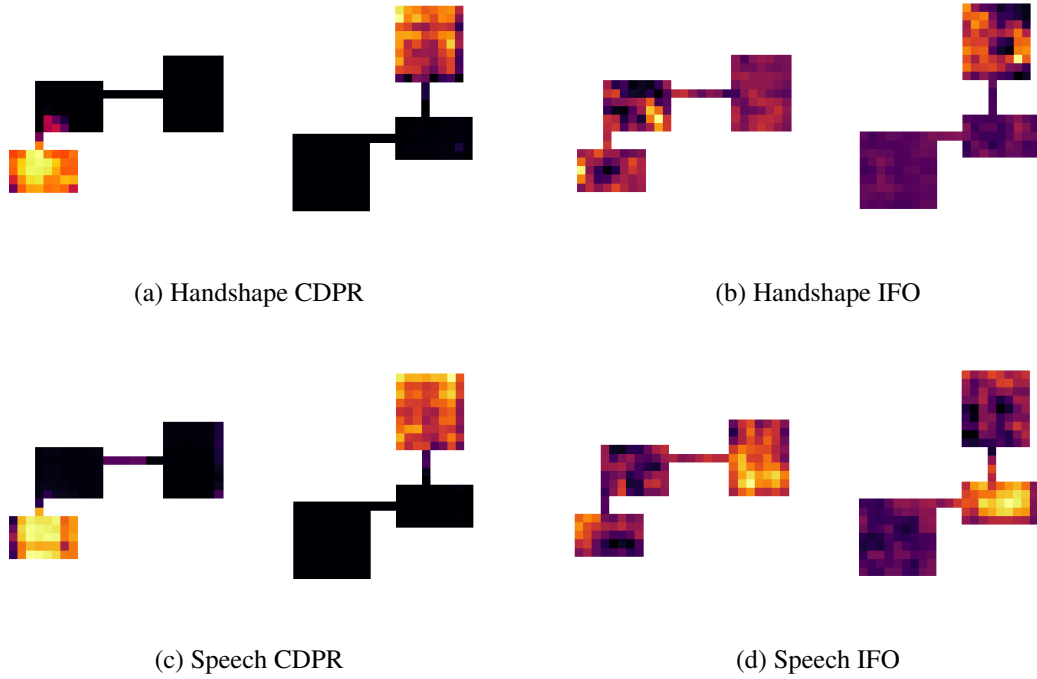


Figure 16: Qualitative results in the Maze domain. To obtain these results, we spawned the agent in each location of the maze, and then produced a heat map of the rewards received. The results are best seen in color.

In our experiments, we aimed to answer the following questions:

1. *Do CDPRs learn correct rewards?*
2. *Do CDPRs learn as well as standard approaches?*

**Does CDPR learn correct rewards?** We first give qualitative results for the Maze task. Figure 16 shows a heat map of the rewards obtained from each location the agent could be spawned in. We do not show results for the standard reward, since this would just give rewards of 1 in the correct room and 0 otherwise. It is clear that CDPRs have learned to adequately reward the correct rooms for both the handshape and speech goal specifications. The largest reward values are given when the agent is in the correct room. The IFO formulation however gives incorrect rewards in most of the rooms. Furthermore, each room has some reward, although the rewards for IFO are negative. It is possible that these intermediate rewards may lead to reward hacking.

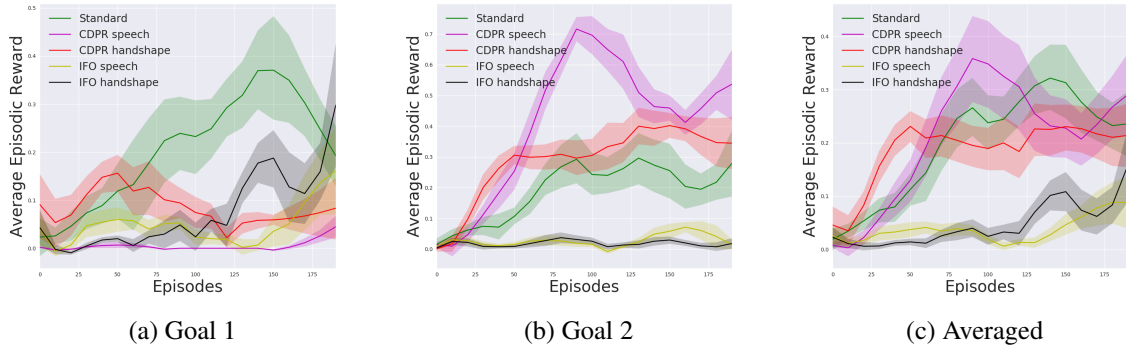


Figure 17: Smoothed results for running RL in the Maze domain. We ran RL with the goal specifications described in Figure 15a.

**Do CDPRs learn as well as standard approaches** Finally, we report the results for using the CDPRs with deep RL, as shown in Figure 17. The desired goals were for the agent to reach the green and yellow rooms, respectively, in the rooms displayed in Figure 15a. On average, both cross-domain representations were able to learn as well as the standard reward approach. It is clear that CDPR can be a suitable replacement for hand-designed reward functions.

IFO only learns how to solve the first goal, where it learned the correct rewards, and on average it performs much worse than the standard approach and CDPRs. Therefore, we have demonstrated that it is more beneficial to learn a reward function directly from cross-domain representations, rather than to generate the corresponding image and use the features for computing a reward, as IFO does.

## 5.6 Conclusion

In this chapter, we have described how we can use CDPRs to learn rewards from goals specified across domains. We have demonstrated the generality of the approach, and have shown that goals from unknown environments can be solved with the learned rewards. Because this approach learns correspondences, it addresses a key problem from the previous chapter, namely that such correspondences can be difficult to tune by hand. We could, for

example, now learn a correspondence between a robot and human face without needing to use an intermediate representation such as a motion template.

To summarize the contributions introduced in this chapter, we have demonstrated that:

1. We can represent goals through images that we choose that are agnostic to the agent's state representation, internal reward, and action space, thus demonstrating that this is as straightforward as domain-specific rewards;
2. CDPRs only needed to be specified once and can be used for unseen tasks, thus demonstrating generality;
3. The agent could perform as well as the standard agent, thus demonstrating that PRFs representation equally enables task completion.

This concludes our discussion of single goal-based specifications for emulation learning. In the next two chapters, we will discuss how to use imitation learning from perceptual goal specifications.

## Chapter VI

### PERCEPTUAL VALUES FROM OBSERVATION

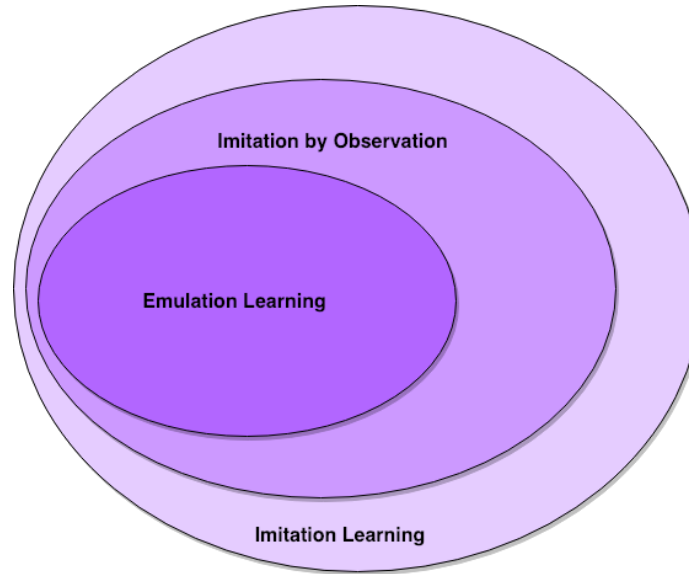


Figure 18: Emulation vs Imitation. There is a trade-off between the amount of information we give the agent in a single demonstration and the amount of experience it will take the agent to learn. Imitation by Observation then offers a middle-ground between environment experience and demonstration data.

The emulation learning approaches discussed in Chapters 4 and 5 offer many advantages over standard representations. They demonstrate how goals can be specified without requiring domain-specific reward engineering and that these goals can be obtained from sources that we choose. Utilizing a single goal offers the additional benefit in that it requires less effort from experts than a full demonstration.

Nevertheless, these approaches are essentially sparse reward problems and do not help to speed up reinforcement learning. While emulation learning, or single-goal representations, require little demonstration data, they need more environment interaction to train reinforcement learning agents. On the other-hand, imitation learning approaches need less environment interaction but more demonstration data from an expert. Figure 18 explains the

trade-off between emulating from a single goal image, and imitating from a sequence of state observations.

What we desire is for agents to be able to learn simply by observing other agents act. This is similar to how humans learn and could enable learning from realistic situations that are not dependent on the agent’s MDP. The key problem for this is that observations do not contain actions that tell the agent how to behave in the world. Furthermore, the action representations might be different from the agent’s. The remaining two chapters discuss how to learn from observations without action information. This dissertation will take two approaches to this problem.

In this chapter, we introduce learning values from observation.

## **6.1 Introduction**

Observational learning is a key component for human development that enables solving tasks by observing others, even when we are unaware of their driving forces and intentions [45]. Often, we can infer states that are good to be in without knowing what the exact solution looks like. For example, when building a piece of furniture, we can typically make an assumption that it is better to have the the pieces outside of the box than in, and a leg screwed into its base than on the floor.

In relation to this, there is typically a clear ordering to which steps are more preferable when completing a problem—final steps are more desirable than earlier ones, assuming the task is being completed optimally, because they indicate that we have fewer steps left to go. Put in other terms, these later steps are typically more *valuable* than those seen in the beginning.

Artificial agents equipped with the ability to infer values solely from observation could quickly learn which actions to take to reach these valuable states, even without knowing the underlying reward function or actions of the observed expert. In imitation learning problems, task goals are often achieved at or near the end of a demonstration trajectory, and so it is



likely that later states have more value than early ones.

In this chapter, we use this insight to compute perceptual *values* from expert observations without access to expert actions. Given expert trajectories, we can compute the expected value of each observation by assuming the reward at the last state of the trajectory is 1 and 0 everywhere else, and then backing this up to the start of the trace. We show how these values can be used to train action-values for reinforcement learning.

There are several advantages to using this approach:

1. It allows learning from a sequence of observations without actions;
2. It learns values that can be used to more quickly train reinforcement learning than sparse rewards.

In this chapter, we introduce Perceptual Values from Observation (PVO). As we will demonstrate, this approach additionally enables learning a value function that is capable of generalizing to task configurations not seen within the training data. We demonstrate the value learning in a maze environment [99] and human pouring dataset [72], and reinforcement learning within OpenAI’s CoinRun environment [20].

## **6.2 Formalities**

We aim to use PVO to learn problems specified through an MDP, where we again do not have access to the transition function or environment rewards, and the states consist of visual inputs, although this is not required for this approach. We are given a set of expert state observations  $D$  where we assume we also do not have access to the underlying expert actions or rewards.

In summary, the assumptions made in this chapter are:

**Assumptions:**

- ◇ States consist of images;
- ◇ The environment reward function is unknown;
- ◇ The transition function is unknown;
- ◇ Expert examples of sequences of state observations.

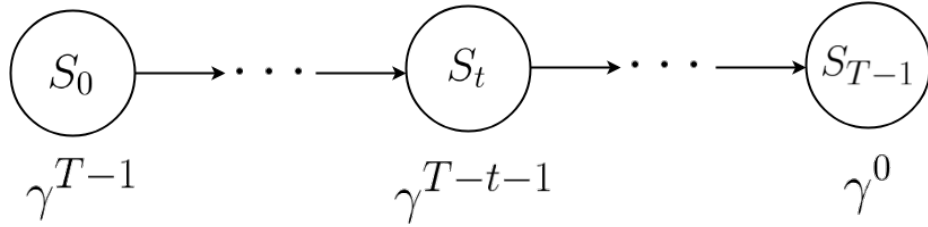
**6.3 Approach**

Figure 19: Value assignment for a length  $T$  trajectory sampled from expert demonstrations.

Given a trajectory of expert observations  $\{s_0^* \dots s_n^* \in D\}$ , we aim to learn a value function  $V_\theta(\cdot)$  that makes an approximation of the expert value function  $V_{\pi_E}(\cdot)$ . Here,  $\pi_E$  is the unobserved expert policy. In this approach, we make the observation that we do not need actions to estimate values of states, but can rather utilize samples from rollouts, i.e., the expert demonstrations.

As we noted, we are not given the underlying reward function with these demonstrations. Rather, we enforce a surrogate reward based on a simple assumption: *Tasks obtained from expert observations can be specified through a sparse reward of 1 at the end of the trajectory and 0 elsewhere.*

This hypothesis comes from the observation that the goal will often occur at the end of the trajectory, especially in goal-directed tasks. However, we enforce this reward function

even if a trajectory does not actually end at the goal. Using this assumption, we may backtrack values from the end of a trajectory to the start without knowing the actions taken. We can then use this value function to learn values of novel states and to learn action-values for reinforcement learning.

### 6.3.1 Step 1: Learning values from observation

---

**Algorithm 2** Perceptual Values from Observation

---

```

1: function PVO( $D$ )
2:   Learn values from observation
3:   for  $k \leftarrow 0 \dots \#Epochs$  do                                 $\triangleright$  (Omitting batching for clarity)
4:      $\langle s_0^*, s_1^*, \dots, s_T^* \rangle \sim D$                                  $\triangleright$  (Sample demonstration)
5:      $t \sim \text{uniform}(0, T)$ 
6:      $y_k = \gamma^{T-t-1}$                                                $\triangleright$  (Compute target value)
7:      $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_{\theta} \|y_k - V_{\theta}(s_t)\|^2$      $\triangleright$  (Update parameters)

```

---

The first step aims to obtain values from expert observations. Given a length  $T$  trajectory  $\{s_0^* \dots s_{T-1}^* \in D\}$ , we first make the assumption that  $s_{T-1}^*$  is a terminal goal state, and so its reward is 1.

The expected value of some state  $s_k$  can be expressed as:

$$V(s_k) = \mathbb{E} \left[ \sum_{t=0}^{T-1} \gamma^t r_{t+k} \right] \quad (30)$$

Because the reward at  $s_{T-1}^*$  is 1, we can assign the values using samples from the demonstration as:

$$V(s_{T-1}^*) := 1, V(s_{T-2}^*) := \gamma, V(s_{T-3}^*) := \gamma^2 \dots, V(s_0^*) := \gamma^{T-1}.$$

In general, we can express the value of some observation  $s_t^*$  as:

$$V(s_t^*) := \gamma^{T-t-1}. \quad (31)$$

This update is shown in figure 19. Here  $T - t - 1$  is effectively the number of steps

remaining in the trajectory. It corresponds to how much the value at the goal will be discounted from state  $t$  before reaching the terminal state. Because we have ground-truth data, we *know* how many steps are left to go.

We learn the values through a purely monte-carlo based approach. However, now the sampled episodes are obtained from the expert and not from an agent's direct experience, as is typical.

In particular, we use a deep neural network to learn the values, and aim to minimize the following loss:

$$L_{\theta} = \|V_{\theta}(s_t) - \gamma^{T-t-1}\|^2. \quad (32)$$

This simple yet effective approach is shown in Algorithm 2.

### 6.3.2 Step 2: Learning action-values from values

Given the learned values, we aim to use reinforcement learning to learn action-values and a corresponding policy. We introduce two approaches to this problem: 1) using the values to replace bootstrapping in Q-learning and 2) using the values as a potential-based shaping reward.

#### 6.3.2.1 Replacing bootstrapping in Q-learning

Recall from Chapter 2 that the typical loss update for Q-learning can be defined as:

$$L_{\theta} = \|y_i - Q_{\theta}(s_t, a_t)\|^2, \quad (33)$$

where  $y_i = r + \gamma \max_a Q_{\theta}(s_{t+1}, a_{t+1})$ . The problem with this approach is that it requires making estimates based off of a moving target. We aim to remove bootstrapping by replacing the target network with our estimate of the value function.

Recall that the maximal action value is equivalent to the value of a state:

$$\max_a Q(s, a) = V(s). \quad (34)$$

Given this definition, we can replace the max from equation 33 with the learned value function  $V_\theta$ , and modify the target accordingly:

$$y_i = r + \gamma V_\theta(s_{t+1}) \quad (35)$$

Because we assume a sparse reward obtained only at the goal, and because we do not compute action-values at terminal states,  $r$  can be replaced with the surrogate reward of 0, and so the target becomes:

$$y_i = \gamma V_\theta(s_{t+1}). \quad (36)$$

#### 6.3.2.2 *Potential-based shaping reward*

If the value function is incorrect for some states, using it as a replacement for bootstrapping might be too strong of a signal. That is because the formulation aims to directly maximize the value function, and so may get stuck in locally sub-optimal areas.

As such, we also introduce using a potential-based shaping reward [56]:

$$r = \gamma V_\theta(s_{t+1}) - V_\theta(s_t). \quad (37)$$

## 6.4 *Experiments*

Our experiments aim to demonstrate that PVO can learn values from observation only and that these values can be used to train reinforcement learning agents. We evaluate the agent within unseen environments and expect to find that PVO learns a general value function that can infer values outside of the training environments. Training details can be found in the appendix.

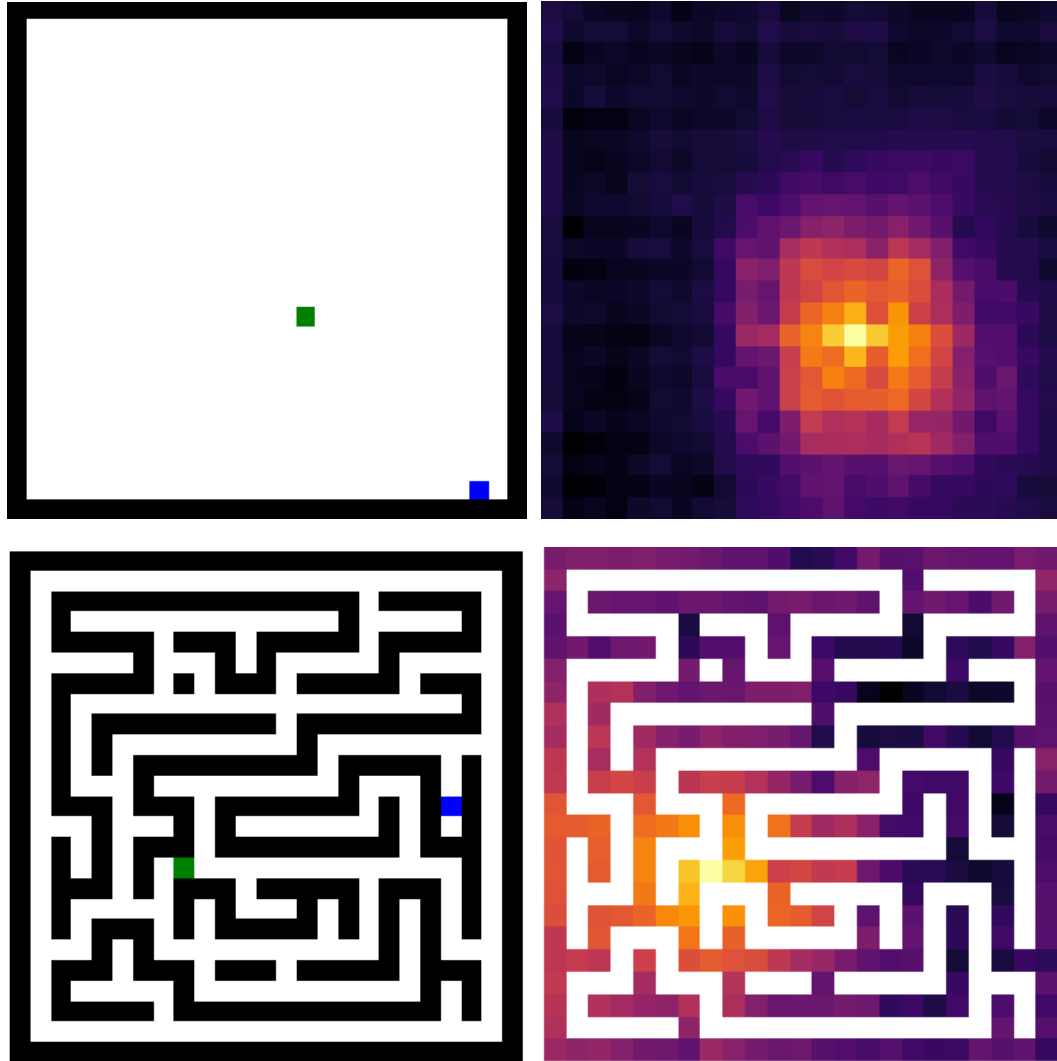


Figure 20: Heatmap of values learned by PVO in unseen maze environments. Brighter colors have larger values. The results are best seen in color.

#### 6.4.0.1 *Environments*

In this section, we discuss the environments used for evaluation. We were interested in goal-directed tasks that aimed to reach a desired target state. We were additionally interested in demonstrating generalization. We used procedurally generated environments, which are randomly generated, and so can show if PVO was capable of generalization.

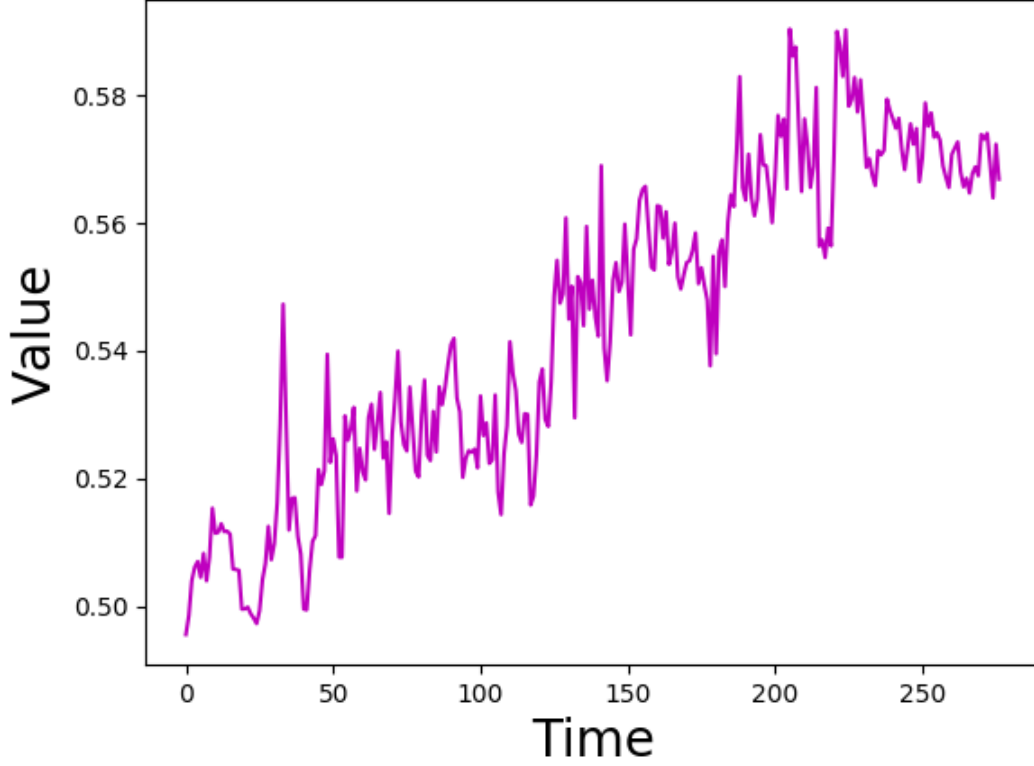


Figure 21: Values learned by PVO in the pouring dataset. The top row represents frames from a single, unseen video. The bottom row represents the learned values for each frame in the video.

#### 6.4.0.2 Maze environment

The maze environment, shown in figure 20, consists of procedurally generated mazes. The agent can take actions up, down, left, and right. The game ends when the agent (blue) reaches some target goal (green). We used  $A^*$  search to obtain demonstrations in this environment. The demonstration set only consisted of mazes from sizes  $4 \times 4$  to  $20 \times 20$ . We aim to determine if PVO can learn values in unseen mazes of size  $25 \times 25$ . We obtained 1000 episodes of demonstrations for a simple empty maze and a more complicated one where the

agent must navigate around obstacles to reach the goal.

#### 6.4.0.3 *Pouring dataset*

The pouring dataset has been used to train robots to learn to pour from videos of humans [72]. We use 10 pouring demonstrations to train values and aim to determine if PVO can infer values on in an unseen video.

#### 6.4.0.4 *CoinRun environment*

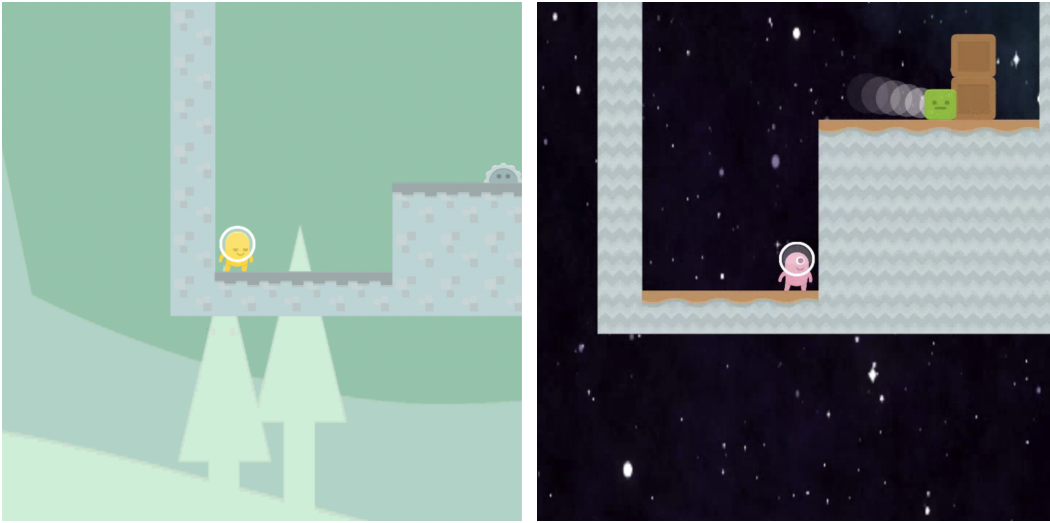


Figure 22: Example of a procedurally generated levels in CoinRun. In this environment, an agent needs to reach a single coin at the end of a platform.

The CoinRun environment consists of procedurally generated platform environments. In particular, the background, player, enemies, platforms, obstacles, and goal locations are all randomly instantiated. The agent can take actions left, right, jump, and down, jump-left, jump-right, and do-nothing. The game ends when the agent reaches a single coin in the game. We trained PPO [70] for 2.5 million steps to obtain 1000 episodes of expert demonstrations. We evaluate on unseen levels.



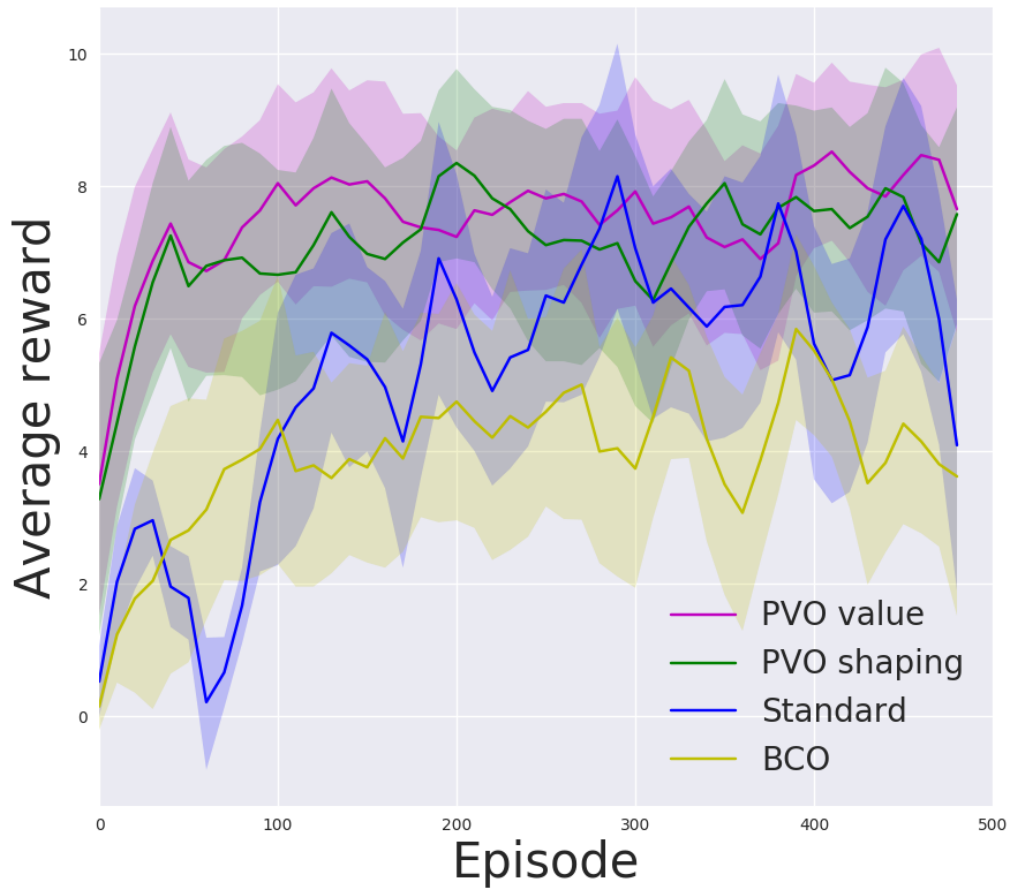


Figure 23: CoinRun results. The trials were averaged over 5 runs with a different, unseen, procedurally generated level for each method. The policy was evaluated for 10 runs every 10 steps. The evaluation metric was a negative step cost.

#### 6.4.1 Results

In this section, we discuss the results of our approach. Our experiments aim to answer the following questions:

1. *Does PVO learn meaningful values?*
2. *Can PVO be used for training reinforcement learning?*

**Does PVO learn meaningful values?** Our experiments in the maze environment aim to demonstrate that PVO can learn meaningful values in unseen environments. Figure 20

shows a heatmap of the values learned using this approach. It is clear that not only is PVO capable of detecting where the goal is, it can also infer the values of states around the goal. The values surrounding the goal as the agent gets further away from it.

We also demonstrate value learning in the pouring dataset, as shown in figure 21. PVO has clearly learned a meaningful value function for this task, even though it was only trained with 10 demonstrations. The initial image is an empty glass without any pouring and the value is clearly low. As time increases and the glass gets fuller, the values increase.

**Can PVO be used for training reinforcement learning agents?** Our experiments in the CoinRun environment aim to demonstrate that PVO can be used for training reinforcement learning agents in unseen environments. We compare against Behavioral Cloning from Observation (BCO) [89], which uses learned inverse dynamics in the agent’s environment to approximate a policy from state observations. Such an approach may have difficulty generalizing as the dynamics in the agent’s environment may look different than the dynamics in the demonstration set.

The results are shown in figure 23. We call the PVO method that replaces bootstrapping with the learned values PVO value and the method that uses the values as a shaping reward PVO shaping.

Both PVO methods learn significantly faster than standard RL. Therefore, we have demonstrated that PVO can be used to speed up learning and can generalize to unseen environments after receiving observation data only. This demonstrates that PVO can be used to replace bootstrapping for RL, but that the shaping reward was also powerful enough that it may not be necessary. One reason for this may be that using the value function to replace bootstrapping essentially initializes the Q-values, which has been shown to be equivalent to potential-based reward shaping [93].

Additionally, even though we make the assumption that trajectories end at the goal, in practice this is not always true. The expert data that we use in CoinRun only had a 67% success rate, and yet our approach was able to learn. As such, we have also demonstrated

that our approach is robust to imperfect demonstrations that do not all end at the goal.

## **6.5 Discussion and Conclusion**

We have demonstrated that PVO works across a diverse set of tasks, is robust to noisy and sub-optimal demonstrations, and can generalize to unseen configurations. We have shown that this can significantly speed up reinforcement learning within sparse reward settings.

There are other reasons why this is useful. It essentially gives an estimate of how close the agent is to completing the task. Additionally, it could be beneficial to know how valuable a state is to be in.

To summarize the contributions introduced in this chapter, we have demonstrated that:

1. We can represent goals through images that we choose that are agnostic to the agent's internal reward, and action space, thus demonstrating that this is as straightforward as domain-specific rewards;
2. Learned values from PVOs can be used for unseen goals, thus demonstrating generality;
3. The agent could perform better than the standard agent, thus demonstrating that PRFs representation equally enables task completion.

One limitation of this approach is that it required using reinforcement learning, thus it needed to learn an action-value function in order to obtain a policy. In the next chapter, we will describe how we can learn policies directly from observation.

## Chapter VII

### IMITATING LATENT POLICIES FROM OBSERVATION

This chapter introduces the final approach used in this dissertation. Unlike the emulation learning approaches, PVO was designed to speed up reinforcement learning. However, each of these approaches still involve learning action-values, which requires much interaction with the environment. Furthermore, the approaches required defining a reward function, which may still be susceptible to specification problems. The approach described in this chapter aims to learn a policy directly from observation, and eliminates the need to use reinforcement learning at all.

In particular, we will introduce learning latent policies from observation. This is a powerful method because once a latent policy is learned, true policy learning can be done quickly. We will demonstrate the approach in classic control environments and CoinRun.

The code for this work is available at <https://github.com/ashedwards/ILPO>.

#### *7.1 Introduction*

In this chapter, we describe a novel approach to imitation learning that infers latent policies directly from state observations. We introduce a method that characterizes the causal effects of unknown actions on observations while simultaneously predicting their likelihood. We then outline an action alignment procedure that leverages a small amount of environment interactions to determine a mapping between latent and real-world actions. We show that this corrected labeling can be used for imitating the observed behavior, even though no expert actions are given. We evaluate our approach within classic control and photo-realistic visual environments and demonstrate that it performs well when compared to standard approaches.

Humans often learn from and develop experiences through mimicry. Notably, we are capable of mirroring behavior through only the observation of state trajectories without

direct access to the underlying actions (*e.g.*, the exact kinematic forces) and intentions that yielded them [62]. In order to be general, artificial agents should also be equipped with similar forms of mimicry; however, imitation learning approaches typically require both observations and actions to learn policies along with extensive interaction with the environment.

A recent approach for overcoming these issues would be to learn an initial self-supervised model for *how* to imitate by collecting experiences within the environment and then using this learned model to infer policies from expert observations [59, 89]. However, unguided exploration can be risky in many real-world scenarios and costly to obtain. Thus, we need a mechanism for learning policies from observation alone without requiring access to expert actions *and* with only a few interactions within the environment.

In order to tackle this challenge, we hypothesize that predictable, though unknown, causes may describe the classes of transitions that we observe. These causes could be natural phenomena in the world, or the consequences of the actions that the agent takes. This work aims to demonstrate how an agent can predict and then imitate these latent causes, even though the ground truth environmental actions are unknown.

We follow a two-step approach, where the agent first learns a policy offline in a latent space that best describes the observed transitions. Then it takes a limited number of steps in the environment to ground this latent policy to the true action labels. We liken this to learning to play a video game by observing a friend play first, and then attempting to play it ourselves. By observing, we can learn the goal of the game and the types of actions we should be taking, but some interaction may be required to learn the correct mapping of controls on the joystick.

We first make the assumption that the transitions between states can be described through a discrete set of latent actions. We then learn a forward dynamics model that, given a state and latent action, predicts the next state and prior, supervised only by {state, next state} pairs. We use this model to greedily select the latent action that leads to the most probable

next state. Because these latent actions are initially mislabeled, we use a few interactions with the environment to learn a relabeling that outputs the probability of the true action.

We evaluate our approach in three environments: classic control with cartpole and acrobot, and a recent platform game by OpenAI, CoinRun [20]. We show that our approach is able to perform as well as the expert after just a few steps of interacting with the environment, and performs better than a recent approach for imitating from observations, Behavioral Cloning from Observation [89].

## 7.2 Approach

We now describe our approach, Imitating Latent Policies from Observation (ILPO), where we train an agent to imitate behaviors from expert state observations.

---

### Algorithm 3 Imitating Latent Policies from Observation

---

```

1: function ILPO( $s_0^*, s_1^*, \dots, s_N^*$ )
2:   Step 1: Learning latent policies
3:   for  $k \leftarrow 0 \dots \#Epochs$  do
4:     for  $i \leftarrow 0 \dots N - 1$  do  $\triangleright$  (Omitting batching for clarity)
5:       Train latent dynamics  $\theta \leftarrow \theta - \nabla_{\theta} \min_z \|G_{\theta}(E_{p\theta}(s_i^*), z) - s_{i+1}^*\|_2^2$ 
6:       Train latent policy  $\omega \leftarrow \omega - \nabla_{\omega} \|\sum_z \pi_{\omega}(z|s_i^*) G_{\theta}(E_{p\theta}(s_i^*), z) - s_{i+1}^*\|_2^2$ 

7:   Step 2: Action remapping
8:   Observe state  $s_0$ 
9:   for  $t \leftarrow 0 \dots \#Interactions$  do
10:    Choose latent action  $z_t \leftarrow \arg \max_z \pi_{\omega}(z|E_{a\xi}(s_t))$ 
11:    Take  $\varepsilon$ -greedy action  $a_t \leftarrow \arg \max_a \pi_{\xi}(a|z_t, E_{a\xi}(s_t))$ 
12:    Observe state  $s_{t+1}$ 
13:    Infer closest latent action  $z_t = \arg \min_z \|E_{p\theta}(s_{t+1}) - E_{p\theta}(G_{\theta}(E_{p\theta}(s_t), z))\|_2$ 
14:    Train action remapping parameters  $\xi \leftarrow \xi + \nabla_{\xi} \log \frac{\pi_{\xi}(a_t|z_t, E_{a\xi}(s_t))}{\sum_a \pi_{\xi}(a|z_t, E_{a\xi}(s_t))}$ 

```

---

### 7.2.1 Problem formulation

We aim to use ILPO to solve problems specified through an MDP [84], where the reward function and transition dynamics are unknown. We show results for both image and vector-based state inputs. In this work, we use imitation learning to directly learn a policy and

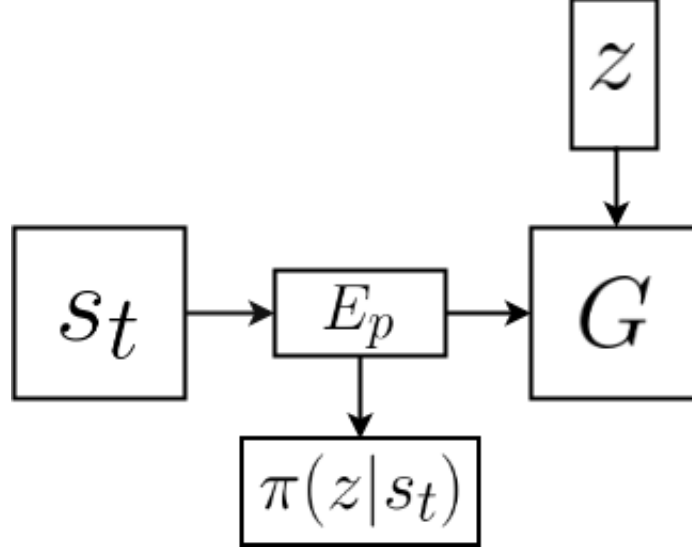
use the reward only for evaluation purposes. We use imitation learning to directly learn the policy and use the reward  $r_t$  only for evaluation purposes.

We are given a set of expert demonstrations described through noisy state observations  $\{s_1^* \dots s_n^*\} \in D$ . In our approach, we will use these observations to predict a multimodal forward dynamics model. As such, noise is necessary for ensuring that state transitions are properly modeled.

Given two consecutive observations  $\{s_t, s_{t+1}\}$ , we define  $z$  as a *latent action* that caused this transition to occur. As such, the action spaces that we consider are discrete with deterministic transitions. Because our problems are specified through MDPs, we assume that the number of actions,  $|A|$ , is known. Hence, we can define  $\{z_1 \dots z_{|A|}\} \in Z$  latent actions, where  $|Z| = |A|$  is used as an initial guess for the number of latent actions. However, there may be more or less types of transitions that appear in the demonstration data. For example, if an agent has an action to move left but always moves right, then the "left" transition will not be observed. Or if the agent moves right and bumps into a wall, this stationary transition may appear to be another type of action. As such, we will empirically study the effect of using latent actions when  $|Z| \neq |A|$ .

**Assumptions:**

- ◇ States consist of images or vectors;
- ◇ The environment reward function is unknown;
- ◇ The transition function is unknown;
- ◇ Expert examples of sequences of state observations.



(a) Latent Policy Network

Figure 24: The latent policy network learns a latent policy,  $\pi(z|s)$ , and a forward dynamics model,  $G$ .

### 7.2.2 Step 1: Learning latent policies

In perhaps the most straightforward approach for imitation learning, behavioral cloning, given expert states and actions  $\{s_1, a_1 \dots s_n, a_n\}$ , we can use supervised learning to approximate  $\pi(a|s_t)$ . That is, given a state  $s_t$ , this approach predicts the probability of taking each action, i.e., the policy. However, imitation by observation approaches do not have access to expert actions. To address this, behavioral cloning from observation (BCO) [89] first learns an inverse dynamics model  $f(a|s_t, s_{t+1})$  by first collecting samples in the agent's environment. Then, the approach uses this model to label the expert observations and learn  $\pi(a|s_t)$ . However learning dynamics models online can require a large amount of data, especially in high-dimensional problems.

We make the observation that we do not need to know action labels to make an initial hypothesis of the policy. Rather, our approach aims to learn a *latent* policy  $\pi_\omega(z|s_t)$  that estimates the probability that a latent action  $z$  would be taken when observing  $s_t$ . This process can be done *offline* and hence more efficiently utilizes the demonstration data.

In order to learn this latent policy, we introduce a latent policy network with two key



components: a latent forward dynamics model  $G$  that learns to predict  $\hat{s}_{t+1}$ , and a prior over  $z$  given  $s_t$ , which gives us the latent policy, as shown in figure 24. We then use a limited number of interactions with the environment to learn an action-remapping network that efficiently associates the true actions the agent can take with the latent policy identified by our learned model. These methods are outlined in algorithm 3 and will be discussed in the remainder of this section.

Finally, rather than operating directly on state inputs, we use an embedding to encode the states. This is useful for high-dimensional inputs, and additionally for reuse within different components of the network. We use a neural network to learn an embedding  $E_p$  concurrently with the latent policy and forward dynamics networks. We additionally learn another embedding  $E_a$  for the action remapping network.

#### 7.2.2.1 Latent forward dynamics

We now describe how to learn a latent forward dynamics model from expert state observations. Given an expert state  $s_t$  and latent action  $z$ , our approach trains a generative model  $G_\theta(E_p(s_t), z)$  to predict the next state  $s_{t+1}$ . Similar to recent works that predict state dynamics [27, 34], our approach predicts the differences between states  $\Delta_t = s_{t+1} - s_t$ , rather than the absolute next state, and computes  $s_{t+1} = s_t + \Delta_t$ .

When learning to predict forward dynamics, a single prediction,  $f(s_{t+1}|s_t)$ , will not account for the different modes of the distribution, i.e., the effects of each action, and will thus predict the mean over all transitions. When using an action-conditional model [17, 58], learning each mode is straightforward, as we can simply make predictions based on the observed next state after taking each action,  $f(s_{t+1}|s_t, a)$ . However, in our approach, we do not know the ground truth actions that yielded a transition. Instead, our approach trains a generative model  $G$  to make predictions based on each of the latent actions  $z \in Z$ ,  $f(s_{t+1}|s_t, z)$ .

To train  $G$ , we compute the loss as:

$$L_{min} = \min_z \|\Delta_t - G_\theta(E_p(s_t), z)\|^2. \quad (38)$$

To allow predictions to converge to the different modes, we only penalize the one closest to the true next observation,  $s_{t+1}$ . Hence the generator must learn to predict the closest mode within the multi-modal distribution. This approach essentially allows each generator to learn *transition clusters* for each type of transition that is represented through  $\Delta_t$ . If we penalized each of the next state predictions simultaneously, the generator would learn to always predict the *expected* next state, rather than each distinct state observed after taking a latent action  $z$ .

We use  $\Delta_t$  to better guide the generator to learn these distinct types of transitions. For example, if we have an agent moving in discrete steps within the  $x$ -plane, then moving right would yield positive transitions  $\Delta = 1$  and moving left would yield negative transitions  $\Delta = -1$ . Our approach aims to train the generator to learn these different types of transitions.

Note that since  $G$  is learning to predict  $\Delta_t$ , we will need to add each prediction to  $s_t$  in order to obtain a prediction for  $s_{t+1}$ . For simplicity, in further discussion we will refer to  $G$  directly as the predictions summed with the state input  $s_t$ .

#### 7.2.2.2 Latent policy learning

Crucially, ILPO concurrently learns the latent policy  $\pi_\omega(z|E_p(s_t))$ . This represents the probability that given a state  $s_t$ , a latent transition of the type  $z$  will be observed in the expert data. We train this by computing the expectation of the generated predictions under this distribution, i.e., the expected next state, as:

$$\hat{s}_{t+1} = \mathbb{E}_{\pi_\omega}[s_{t+1}|s_t] \quad (39)$$

$$= \sum_z \pi_\omega(z|s_t) G_\theta(E_p(s_t), z). \quad (40)$$

We then minimize the loss as:

$$L_{exp} = \|s_{t+1} - \hat{s}_{t+1}\|^2 \quad (41)$$

while holding the individual predictions fixed. This approach predicts the probability of each transition occurring. In other terms, this determines the most likely transition cluster.

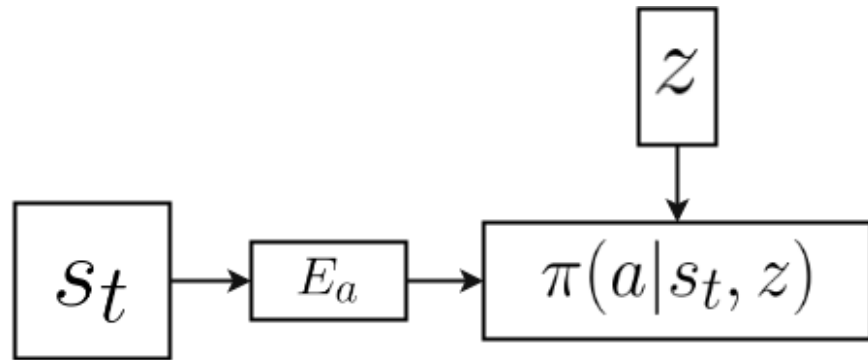
With this loss, the latent policy is encouraged to make predictions that yield the most likely next state. For example, if an agent always moves right, then we should expect, given some state, for the next state to reflect the agent moving right. As such, any other type of transition should have a low probability so that it is not depicted within the next state.

The entire latent policy network is trained using the combined loss:

$$L_{policy} = L_{min} + L_{exp}. \quad (42)$$

We outline the training procedure for this step in lines 3 – 6 in algorithm 1.

### 7.2.3 Step 2: Action Remapping



(a) Action Remapping Network

Figure 25: The action remapping network learns  $\pi(a|s_t, z)$  to align the latent actions  $z$  with ground-truth actions  $a$ .

In order to imitate from expert observations, the agent needs to learn a mapping from the latent policy learned in the previous step to the true action space:  $\pi_{\xi}(a_t|z, E_a(s_t))$ , as

described in figure 25. As such, it is invariantly necessary for the agent to explore the effect of its own actions within its environment. However, unlike BCO and other imitation from observation approaches, ILPO only needs to learn a mapping from  $a$  to  $z$  rather than a full dynamics model. The mapping  $\pi_\xi$  also depends on the current state  $s_t$  because latent actions are not necessarily invariant across states. That is, the actions being predicted by each generator might change in different parts of the state-space. If an agent is flipped upside-down, for example, then the action "move up" would then look like "move down".

Nevertheless, generalization capabilities of neural networks should encourage a strong correlation between  $a$  and  $z$ . The dynamics in two states are often more similar for the same action than they are for two different ones, thus assigning the latent actions to the same type of transition should allow the network to generalize more easily. This intuition will allow us to learn such a mapping from only a few interactions with the environment, but is not a requirement for learning, and the algorithm will be able to learn to imitate the expert's policy regardless.

#### 7.2.3.1 *Collecting experience*

To obtain training data for the remapped policy  $\pi_\xi$ , we allow the agent to interact with the environment to collect experiences in the form of  $\{s_t, a_t, s_{t+1}\}$  triples. This interaction can follow any policy, such as a random policy or one that is updated in an online way. The only stipulation is that a diverse section of the state space is explored to facilitate generalization. We choose to iteratively refine the remapped policy  $\pi_\xi$  and collect experiences by following this current estimate, in addition to an  $\epsilon$ -greedy exploration strategy.

#### 7.2.3.2 *Aligning actions*

While collecting experiences  $\{s_t, a, s_{t+1}\}$  in the agent's environment, we proceed in two steps to train the remapped policy. First, we identify the latent action that corresponds to the environmental state transitions  $\{s_t, s_{t+1}\}$  and then we use the environmental action  $a$  taken as a label to train  $\pi_\xi(a_t|z_t, E_a(s_t))$  in a supervised manner.

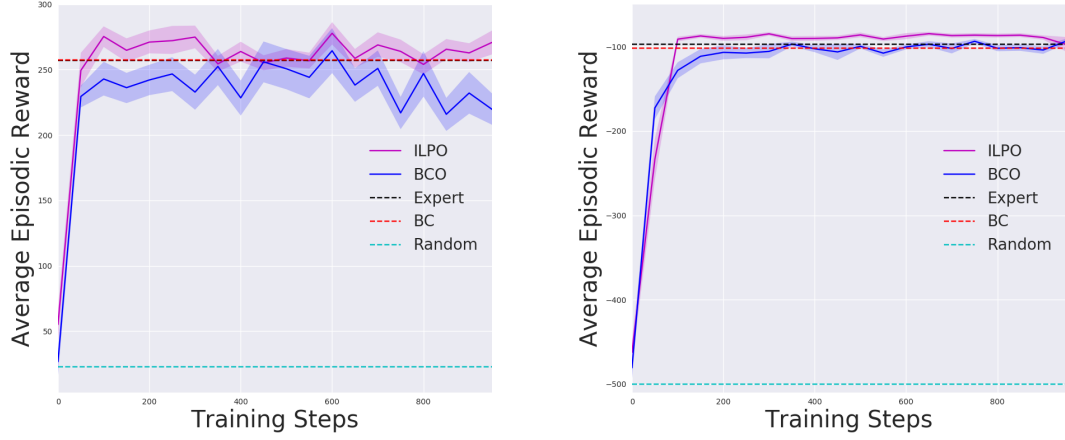


Figure 26: Cartpole and Acrobot results. The trials were averaged over 50 runs for ILPO and the policy was evaluated every 50 steps. The reward used for training the expert and evaluation was +1 for every step that the pole was upright in cartpole, and a -1 step cost for acrobot.

To do this, given state  $s_t$ , our method uses the latent dynamics model,  $G$ , trained in step 1, to predict each possible next state  $\hat{s}_{t+1}$  after taking a latent action  $z$ . Then it identifies the latent action that corresponds to the predicted next state that is the most similar to the observed next state  $s_{t+1}$ :

$$z_t = \arg \min_z \|s_{t+1} - G_\theta(E_p(s_t), z)\|_2. \quad (43)$$

To extend this approach to situations where euclidean distance is not meaningful (such as high-dimensional visual domains), we may also measure distance in the space of the embedding  $E_p$  learned in the previous step. In these domains, the latent action is thus given by:

$$z_t = \arg \min_z \|E_p(s_{t+1}) - E_p(G_\theta(E_p(s_t), z))\|_2. \quad (44)$$

Having obtained the latent actions  $z_t$  most closely corresponding to the environmental action  $a_t$ , we then train  $\pi(a_t|z, s_t)$  as a straight forward classification problem using a cross-entropy

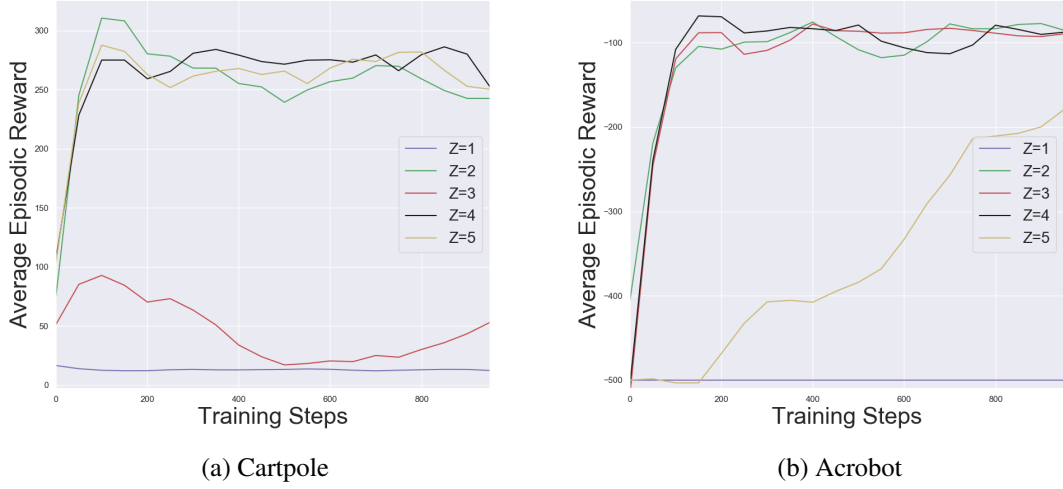


Figure 27: Cartpole and Acrobot results for selecting  $|Z|$ . The trials were averaged over 5 runs for ILPO and the policy was evaluated every 50 steps and averaged out of 10 policy runs. The reward used for training the expert and evaluation was +1 for every step that the pole was upright in cartpole, and a -1 step cost for acrobot.

loss:

$$L_{map} = \log \frac{\pi_{\xi}(a_t | z_t, E_a(s_t))}{\sum_a \pi_{\xi}(a | z_t, E_a(s_t))}. \quad (45)$$

### 7.2.3.3 Imitating latent policies from observation

Combining the two steps into a full imitation learning algorithm, given a state  $s_t$ , we use the latent policy outlined in step 1 to identify the latent cause that is most likely to have the effect that the expert intended,  $z^* = \arg \max_z \pi_{\omega}(z | s_t)$ , and subsequently identify the action that is most likely to cause this effect,  $a^* = \arg \max_a \pi_{\xi}(a | z^*, s_t)$ . The agent can then follow this policy to imitate the expert’s behavior without having seen any expert actions. We outline the training procedure for this step in lines 7 – 14 in algorithm 1.

## 7.3 Experiments and results

In this section, we discuss the experiments used to evaluate ILPO. We aim to demonstrate that our approach is able to imitate from state observations only and with little interactions

with the environment. In addition to this, we aim to show that learning dynamics online through environment interactions is less sample efficient than learning a latent policy first.

We evaluate ILPO within classic control problems as well as a more complex visual domain. We used OpenAI Baselines [24] to obtain expert policies and generate demonstrations for each environment. We compare ILPO against this expert, a random policy, and Behavioral Cloning (BC), which is given ground truth actions, averaged over 50 trials, and Behavioral Cloning from Observation (BCO). More experiment details can be found in the appendix.

### 7.3.1 Classic control environments



Figure 28: CoinRun environment used in experiments. CoinRun consists of procedurally generated levels and the goal is to get a single coin at the end of a platform. We used an easy level (left) and hard level (middle and right) in our experiments. The middle image is the agent’s state observation in the hard level and the image on the right is a zoomed out version of the environment. This task is difficult because the gap in the middle of the platform can not be recovered from and there is a trap as soon as the agent reaches the other side. When training, the images also include a block showing  $x$  and  $y$  velocities.

We first evaluated our approach within classic control environments. We used the standard distance metric from equation 43 to compute the distances between observed and predicted next states for ILPO. We used the same network structure and hyperparameters across both domains, as described in the appendix. We used 50,000 expert state observations to train ILPO and BCO, and the corresponding actions to train Behavioral Cloning (BC).

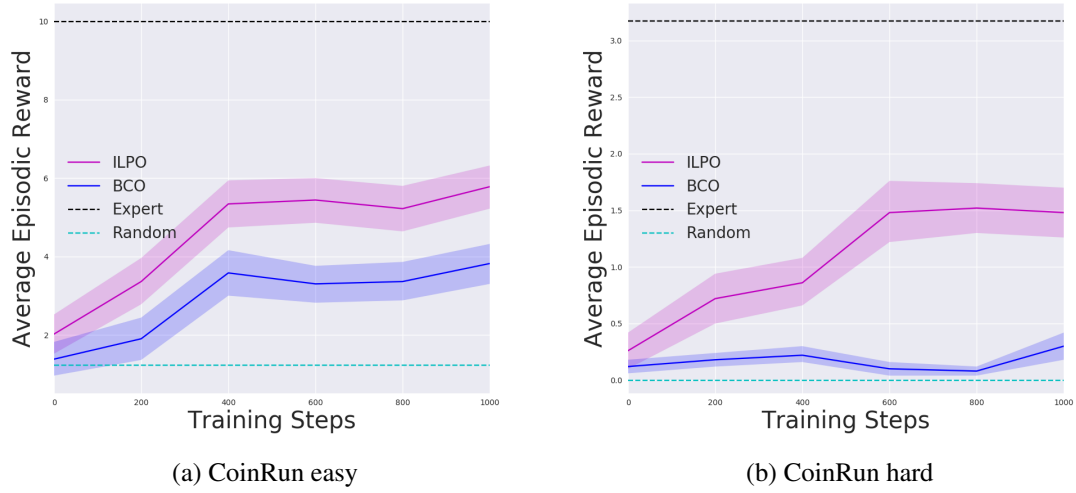


Figure 29: CoinRun imitation learning results. The trials were averaged over 50 runs for ILPO and BCO the policy was evaluated every 200 steps and averaged out of 10 policy runs. The reward used for training the expert and evaluation was +10 after reaching the coin.

### 7.3.2 Cartpole

Cartpole is a classic control environment used in reinforcement learning settings [84]. In this environment, an agent must learn to balance a pole on a cart by applying forces of  $-1$  and  $1$  on it. The state space consists of 4 dimensions:  $\{x, \dot{x}, \theta, \dot{\theta}\}$ , and the action space consists of the 2 forces. As such, ILPO must predict 2 latent actions and generate predicted next states with 4 dimensions.

### 7.3.3 Acrobot

Acrobot is another classic environment, where an agent with 2 links must learn to swing its end-effector up by applying a torque of  $-1$ ,  $0$ , or  $1$  to its joint. The state space consists of 6 dimensions:  $\{\cos \theta_1, \sin \theta_1, \cos \theta_2, \sin \theta_2, \dot{\theta}_1, \dot{\theta}_2\}$ , and the action space consists of the 3 forces. As such, ILPO must predict 3 latent actions and generate a predicted next state with 6 dimensions.



### 7.3.4 Results

Figure 26 (left) shows the imitation learning results in cartpole. ILPO learns the correct policy and is able achieve the same performance as the expert and behavioral cloning in less than 100 steps within the environment. Furthermore, ILPO performs much better than BCO, as it does not need to learn state-transitions while collecting experience, only a mapping from latent to real actions.

Figure 26 shows the imitation learning results in acrobot. ILPO again learns the correct policy after a few steps and is able achieve as good of performance as the expert and behavioral cloning, again within 100 steps. While BCO also learns quickly, ILPO again performs better than this approach.

We were also interested in evaluating the effect of using a different number of latent actions. These results are shown in figure 27. We see that choosing  $|Z| = |A|$  is a good initial guess for the size of  $Z$ , but the agent is also able to learn from other sizes.  $|Z| = 1$  performed poorly in both acrobot and cartpole. This is because every action will collapse to a single action and will not be able to be disentangled.

### 7.3.5 CoinRun

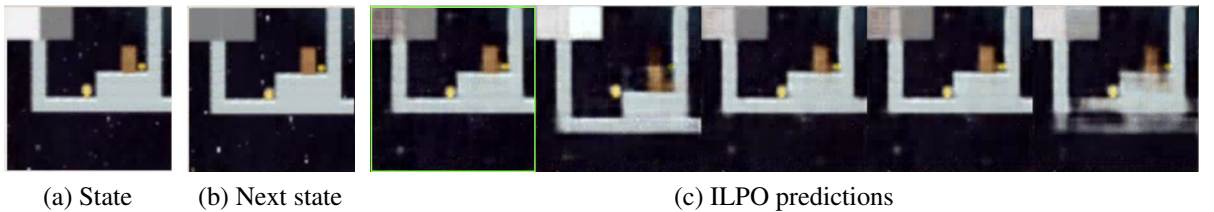


Figure 30: Next state predictions computed by ILPO in the CoinRun easy task. The highlighted state represents the closest next state obtained from equation 44.

We also evaluated our approach in a more complex visual environment, CoinRun [20], which we described in the previous chapter. We used 1000 episodes of expert demonstrations to train ILPO and BCO. In these experiments, we evaluated each approach within a single easy and single hard level, shown in figure 28. This environment is more difficult than

acrobot and cartpole because it uses images as inputs and contains more actions. As such, the dynamics learning takes place over many more dimensions.

In particular, the state space consists of  $128 \times 128 \times 3$  pixels and 7 actions. As such  $128 \times 128 \times 3$  dimensions for the next-state predictions. We found that ILPO performed better when predicting  $|Z| = 5$  latent actions. This is likely because certain actions, such as moving left, were used less frequently. We use the embedded distance metric from equation 44 to compute the distances between observed and predicted next states.

### 7.3.6 Results

Figure 29 shows the results for imitation learning. In both the easy and hard tasks, ILPO was not able to perform as well as the expert, but performed significantly better than BCO. As this environment is high-dimensional, it takes more steps to learn the alignment policy than the previous experiments. However, ILPO often learned to solve the task almost immediately, but some random seeds led to bad initialization that resulted in the agent not learning at all. However, good initialization sometimes allows the agent to learn good initial policies zero-shot. As such, we found that it was possible for the agent to sometimes perform as well as the expert. The results consist of all of the seeds averaged, including those that yielded poor results.

Figure 30 shows the predictions made by the model. ILPO is able to predict moving right and jumping. Because these are the most likely modes in the data, the other generators also predict different velocities. The distance metric is able to correctly select the closest state.

In general, it can often be difficult to learn dynamics from visual inputs. Unlike BCO, by learning a latent policy first, ILPO is able to reduce the number of environment interactions necessary to learn. BCO requires solving both an inverse dynamics model and behavioral cloning each time it collects a batch of experience from the environment. As such, this approach is less efficient than ILPO and in many scenarios would be difficult to perform in

realistic environments.

## ***7.4 Discussion and conclusion***

In this chapter, we introduced ILPO and described how agents can learn to imitate latent policies from only expert state observations and very few environment interactions. In many real world scenarios unguided exploration in the environment may be very risky but expert observations can be readily made available. Such a method of learning policies directly from observation followed by a small number of action alignment interactions with the environment can be very useful. We demonstrated that this approach recovered the expert behavior in three different domains consisting of classic control and challenging vision based tasks. ILPO requires very few environment interactions compared to leading imitation and reinforcement learning methods, provided by the expert observations.

There are many ways that this work can be extended. First, there are two assumptions in the current formulation of the problem: 1) it requires that actions are discrete and 2) assumes that the state transitions are deterministic. Second, the action alignment step can be made even more efficient by enforcing stronger local consistencies between latent actions and generated predictions across different states. This will drastically reduce the number of samples required to train the action alignment network by decreasing variation between latent and real actions. One interesting thing to note is that sometimes, by random chance, the labels for the latent policy network are initialized to the correct ones. In this case, the task can be solved immediately without any environment interaction necessary! One interesting point of research then could be to have humans give an initial labeling to the generators by observing the outputs.

We hope that this work will introduce opportunities for learning to observe not only from similar agents, but from other agents with different embodiments whose actions are unknown or do not have a known correspondence. Another contribution would be to learn to transfer across different environments.

To summarize the contributions introduced in this chapter, we have demonstrated that:

1. We can represent goals through images that we choose that are agnostic to the agent’s internal reward, and action space, thus demonstrating that this is as straightforward as domain-specific rewards;
2. ILPO did not require a reward to be specified at all, thus demonstrating generality;
3. The agent could perform better than the standard agent, thus demonstrating that PRFs representation equally enables task completion.

In general, our work is complimentary to many of the related approaches we have discussed. Many algorithms rely on behavioral cloning as a pre-training step for more sophisticated approaches. As such, we believe ILPO could also be used for pre-training imitation, without requiring access to expert actions.

## Chapter VIII

### DISCUSSION AND FUTURE WORK

#### ***8.1 Introduction***

In this dissertation, we have discussed several approaches for utilizing perceptual goals within reinforcement and imitation learning. We have demonstrated how perceptual goals can be used for emulating from a single image that is obtained from a different environment than the agent's, and for imitating without access to actions. We have shown that these specifications can be agnostic to certain parts of the agent's environment. In this chapter, we discuss the potential expansions to the approaches introduced.

#### ***8.2 Perceptual rewards***

This dissertation has focused primarily on learning from visual perception. However, this is of course not the only sensory input that influences how humans act. As such, we believe there is a lot of room to expand upon this area.

Haptics, audio, and other types of perceptual rewards could introduce a wealth of possibilities for training agents. For example, we can often *hear* when a goal has been accomplished. When pouring a glass of water, for example, we can hear when the water has reached the top because the pitch changes. For a robot, using vision alone might be difficult for determining when to stop pouring.

#### ***8.3 Imitation from other agents***

The world is inherently multi-agent. As such, it will be necessary for agents to be able to learn to behave by observing others. One difficulty is that the action spaces of these agents will often be unknown or different than our own. We described two approaches for imitating from observation alone. However, we focussed on imitating from observations that

were obtained from an expert demonstrator with the same transitions as the agent. It would be interesting to use these method to learn from other agents, since their action spaces might be different. Dynamics learning like that done in ILPO will be difficult to do directly. Value learning approaches like PVO may make it easier to learn action-agnostic behaviors from observation.

## **8.4 *Safety***

One important topic that we have not touched upon this work is safety concerns. Learning priors about the world offers the potential to remove a lot of burden from the designer, but removing supervision introduces potential for having an incomplete understanding of the problem the agent is solving. This could be dangerous, and it is important to keep interpretability in mind when designing perceptual goals.

## Chapter IX

### CONCLUSION

In this dissertation, we have described how perceptual goals can be used in place of domain-specific reward functions for reinforcement and imitation learning. The aim of this dissertation was to demonstrate that: **Employing perceptual goal specifications for goal-directed tasks: is as straightforward as specifying domain-specific rewards; is a more general representation for tasks; and equally enables task completion.** We have described several approaches to support this statement. In the remainder of this dissertation, we will provide final discussions for each method.

#### 9.1 *Strengths and limitations*

Table 1: Comparison between the methods described in this dissertation: PRFs, CDPRs, PVO, and ILPO.

	Emulation	Imitation	Hand-defined	Learned	Generalization	Cross-domain	Discrete	Continuous	RL
PRF	✓		✓			✓	✓	✓	✓
CDPR	✓			✓	✓	✓	✓	✓	✓
PVO		✓		✓	✓		✓	✓	✓
ILPO		✓		✓			✓		

In this section, we will compare and contrast each method and describe which conditions each should be used in.

##### 9.1.1 Goal specification

In chapter 6, we discussed the differences between emulation and imitation learning. Both PRFs and CDPRs are emulation learning approaches, and allow specifying a single goal. This however means that learning will take longer. Whereas PVO and ILPO are imitation learning approaches, which means learning is faster but requires demonstrations. As such, if obtaining the goal is easier than obtaining a full trajectory, PRFs or CDPRs would be good

methods to use. However, if we have the full trajectory within the agent’s environment then we should use PVO or ILPO.

Both PRFs and CDPRs aimed to allow specifying goals across domains. As such, when we have a way of representing a goal in an environment that is different from the agent’s, one of these methods would be more appropriate than PVO or ILPO, as those approaches were not designed to handle cross-domain goal specifications.

### **9.1.2 Reward type**

PRFs use a hand-defined reward function and correspondence. Because of this, only a single goal sample is needed to specify a problem. However, this representation may require some tuning. CDPRs and PVO each learn a reward function, and ILPO does not use one at all, but they each require many samples to learn from. As such, if there are not many goal samples available, then PRFs can be used to solve a problem with manual tuning, but if samples are available then the other methods would likely be more appropriate.

### **9.1.3 Generalization**

Because PRFs use a hand-defined correspondence, they are not well-equipped to generalize across goals without tuning. CDPRs learned to generalize to unseen environments, but still required a cross-domain goal to be specified. PVO allowed generalizing without any goal-specification in the unseen environment, but ILPO was only designed to learn within the same environment. As such, for emulation learning problems, CDPRs should be used for generalization, whereas PVO should be used for generalizing in imitation learning problems.

### **9.1.4 Action type**

PRFs, CDPRs, and PVO are not dependent on the action-type of the MDP. Because ILPO models latent transitions, it expects the agent’s actions to be discrete. As such, if expert trajectories are available for an agent with continuous actions, PVO would be an appropriate approach to use.



### 9.1.5 Learning mechanism

PRFs, CDPRs, and PVO each defined a reward function to be used for RL. However, these approaches required learning action-values, which can be slow. While, unlike PRFs and CDPRs, ILPO required a trajectory for the goal-specification rather than a single goal, this approach allowed learning a policy directly without requiring learning values or using rewards. As such, ILPO would be a good approach to use if a problem requires efficiency in environment interactions and expert trajectories are available.

## 9.2 *Performance against traditional reward design*

Some methods described in this dissertation performed better than the standard scalar reward mechanism, whereas others only matched the performance. The emulation learning approaches described in this dissertation, PRF and CDPR, were developed to be an alternative to traditional goal design. As such, we did not aim to outperform standard methods. Typically in imitation learning approaches, we desire for an agent to do as well as the expert. This was our goal for ILPO. PVO, however, was developed to generalize to unseen environments. As such, we aimed for this method to perform better than using reinforcement learning from scratch.

## 9.3 *Extending beyond goal-directed tasks*

In this dissertation, we focused on the problem of solving goal-directed tasks. In particular, we studied how an agent could learn from an explicitly defined goal with a clear terminating state. This type of problem is applicable in many settings, as we have shown throughout this dissertation. Nevertheless, there are several tasks that cannot easily be described by a single, terminating goal. Rather, these tasks, which we refer to as continuing tasks, have some goal that needs to be maintained for the lifetime of an agent. In particular, there are goals that should always be followed (such as maintaining balance for a humanoid or staying on-road for a self-driving car), and terminal “anti-goals” that should be always be avoided (such as

falling or crashing). This is similar to the concept of options and constraints in hierarchical reinforcement learning [41], in which agents learn or are given policies for things to do and things not to do. Subramanian et al. [81] even show that such abstractions are followed by people. As such, it seems suitable that we consider these types of goal specifications as well.

To summarize, there are three cases to consider when specifying goals:

- Explicit goals that terminate the state
- Ongoing goals to maintain (options)
- Anti-goals to avoid that terminate upon failure (constraints)

The methods described in this dissertation use the first point for goal specification. As such, in the remaining sections, we will consider how to extend perceptual goals to the latter two items.

### **9.3.1 Options and constraints for perceptual emulation**

In this dissertation, we have assumed that there was a single goal to be followed. In the emulation learning approaches, PRFs and CDPR, this was expressed through a single perceptual goal. To enable learning options for emulation learning, we would need multiple perceptual goals. A straightforward approach would be to have a different perceptual reward for each goal. To learn constraints, we could simply negate the rewards. Nevertheless, one problem to consider is how to learn from multiple rewards. For example, should the rewards be combined into a single reward function or should we learn a different Q-function for each? Bhat et al. [9] discuss several approaches for how to arbitrate between these possibly conflicting goals. However, as the number of goals increases, it may become more difficult to combine these reward functions. As such, it may be more suitable to use imitative approaches to follow options.

### **9.3.2 Options and constraints for perceptual imitation**

In the imitation learning approaches, ILPO and PVO, we assumed the final state in an expert demonstration was the goal. It is trivial to relax this assumption for ILPO, as this method directly learns a policy based on observations. That is, ILPO already learns the states that it should try to reach (i.e. options) and avoid (i.e. constraints). PVO, on the other hand, uses the assumption that the end of a trajectory is a goal and learns a value function that is then used to train reinforcement learning. As such, it should may be able to learn which states to reach and avoid for a given goal-terminated task, but it would be difficult to extend to continuing tasks.

## Appendix A

### A.1 PRF experiment details

We used Deep Q-Learning to train the agents in each experiment. We used the same DQN architecture used in the original paper [54]. We used an Adam Optimizer [46] with the learning rate initialized to .0001 for training. We followed an  $\epsilon$ -greedy approach while learning, with  $\epsilon$  initialized to 1 and decayed over time. The discount factor was set to .99. We evaluate the learned greedy policy after every 100 episodes for each experiment. We now describe the experiments for two different task descriptors—a direct descriptor and a motion template descriptor.

We needed to tune some parameters before running the algorithm. First, we added in 10 images of the agent’s neutral face to its state sequence  $S$ . We found this was a necessary step to ensure the agent’s template did not decay too quickly. We also needed to blur the inputs for both  $T_S$  and  $T_G$  in order to remove noise. We found that tuning the duration parameter  $\delta$  to  $\frac{|G|}{4}$  and  $\frac{|S|}{3}$  worked well. The parameter  $\tau$  was initialized to .1 and iterated by .1 for  $T_S$  at each each time step and by .3 for  $T_G$ . We found that blurring the images in the videos beforehand gave much better motion templates.

In the motion template task, we used different DQN state inputs for the standard reward and motion PRF. The input for the VRF was the agent’s mirror state (Figure 15) and the inputs for the PRF was the agent’s motion template, or  $T_A$ . The cell size we used for the HOG features was .05h.

### A.2 CDPR experiment details

In order to train the CDPRs for the handshake and the speech specifications, we used a momentum optimizer with an initial learning rate of  $10^{-4}$  and momentum value of .9. We used a batch size of 32. The embedding networks for the CDPR have the following architecture: Conv1  $\rightarrow$  maxpool  $\rightarrow$  Conv2  $\rightarrow$  maxpool  $\rightarrow$  FC1  $\rightarrow$  FC2, where Conv1

consists of 64 11x11 filters with stride 4, Conv2 consists of 192 5x5 filters with stride 2, FC1 outputs 400 features, and FC2 outputs 100. Each maxpool consists of a 3x3 filter with stride 2. Conv1, Conv2, and FC1 are each followed by batch normalization [40] and then ELU [19].

To train IFO, we used the same architecture described in the paper with the unmodified code from [https://github.com/wyndwarrior/imitation\\_from\\_observation](https://github.com/wyndwarrior/imitation_from_observation).

### ***A.3 PVO experiment details***

Each of the experiments used the same network architecture for computing values. The value network had the following architecture: Conv1  $\rightarrow$  Conv2  $\rightarrow$  Conv3  $\rightarrow$  Conv4  $\rightarrow$  Conv5  $\rightarrow$  FC1  $\rightarrow$  FC2  $\rightarrow$  FC3  $\rightarrow$  FC4, where Conv1 consisted of 64 16x16 filters, Conv2 consisted of 128 8x8 filters, Conv3 consisted of 256 4x4 filters, Conv4 consisted of 256 2x2 filters, and Conv5 consisted of 256 1x1 filters, each with stride 2 and with all followed by lrelu. Each fully-connected layer had 32 outputs, except the final layer FC4 which had a single output for the value.

We trained the values for 100000 steps with a batch size of 32. We used a discount factor of .99 in our experiments and a learning rate of .00002.

We used the same DQN architecture used in the original paper [54].

### ***A.4 ILPO experiment details***

We now discuss the hyperparameters and architectures used to train ILPO and behavioral cloning, and BCO. We used the Adam Optimizer to train all experiments.

#### ***A.4.1 Cartpole and Acrobot***

##### ***A.4.1.1 ILPO***

In the latent policy network, the embedding architecture for  $E_p$  was:  $FC_{128} \rightarrow lrelu \rightarrow FC_{256}$ . The leak parameter for *lrelu* was .2 for each case. After converting  $z$  to a one-hot, the generator network encodes it with  $FC_{256}$ , concatenates it with  $E_p$  and places it through *lrelu*,

then placed it into the following architecture to compute  $g(s_t, z) : FC_{128} \rightarrow lrelu \rightarrow FC_{dims}$ , where  $dims$  was the number of state dimensions. To compute  $P(s|z)$ , we compute  $lrelu$  for  $E_p$  and then the following architecture:  $FC_{|A|}$ , where  $|A|$  is the number of actions. We trained the network for 1000 epochs with a batch size of 32. The learning rate was .0002.

In the action remapping network, the embedding architecture for  $E_a$  was the same as  $E_p$ . After converting  $z$  to a one-hot, the policy network encodes it with  $FC_{256}$ , concatenates it with  $E_a$  and performs  $lrelu$ , then places it into the following architecture to compute  $P(a|s_t, z) : FC_{64} \rightarrow lrelu \rightarrow FC_{32} \rightarrow FC_{|A|}$ . We trained the network for 1000 steps with a batch size of 32. The learning rate was .002.

#### A.4.1.2 Behavioral cloning

Behavior cloning used the same architecture as  $E_p$  for encoding states, followed by  $lrelu$  and then  $FC_{|A|}$ .

We trained the network for 1000 epochs with a batch size of 32. The learning rate was .0002.

#### A.4.1.3 BCO

BCO used the same architecture as  $E_p$  for encoding states then placed it into the following architecture to compute  $P(a|s_t) : FC_{64} \rightarrow lrelu \rightarrow FC_{32} \rightarrow FC_{|A|}$ . This is the same architecture as ILPO for  $P(a|s_t, z)$  except it does not take in latent actions. To compute inverse dynamics for  $s_t$  and  $s_{t+1}$ , we place each through  $E_p$  and concatenate them, then compute  $lrelu$  for the concatenation and place through  $FC_{|A|}$ .

We trained the network for 1000 iterations with a batch size of 32. At each iteration, we collected 50 steps of experience from the policy and trained the inverse dynamics model for 10000 steps and the policy for 10000 steps. The learning rate was .0002.

Both BCO and ILPO used  $\varepsilon = 0$  for taking random actions while training and evaluating.

## A.4.2 CoinRun

### A.4.2.1 ILPO

In the latent policy network, the embedding architecture for  $E_p$  was:  $Conv_{30} \rightarrow lrelu \rightarrow Conv_{60} \rightarrow lrelu \rightarrow Conv_{120} \rightarrow lrelu \rightarrow Conv_{120} \rightarrow Conv_{120} \rightarrow lrelu \rightarrow Conv_{120}$ . The parameter for  $lrelu$  was .2 for each case. After converting  $z$  to a one-hot, the generator network encodes it with  $FC_{120}$ , concatenates it with  $E_p$  and does  $lrelu$ , then places it into the following architecture to compute  $g(s_t, z) : Deconv_{120} \rightarrow lrelu \rightarrow Deconv_{120} \rightarrow lrelu \rightarrow Deconv_{120} \rightarrow lrelu \rightarrow Deconv_{60} \rightarrow lrelu \rightarrow Deconv_{30} \rightarrow lrelu \rightarrow Deconv_{15}$ . All filters were size 4x4 with stride 2. We trained the network for 10 epochs with a batch size of 100. The learning rate was .0002.

In the action alignment network, the embedding architecture for  $E_a$  was the same as  $E_p$ . After converting  $z$  to a one-hot, the generator network encodes it with  $FC_{256}$ , concatenates it with  $E_p$  and places it through  $lrelu$ , then places it into the following architecture to compute  $P(a|s_t, z) : FC_{64} \rightarrow lrelu \rightarrow FC_{64} \rightarrow FC_{|A|}$ . We trained the network for 1000 steps with a batch size of 100. The learning rate was .001.

### A.4.2.2 BCO

We found that BCO overfit when it used the same architecture as  $E_p$  for encoding states. As such, we first created the state embedding using the following architecture:  $Conv_{15} \rightarrow lrelu \rightarrow Conv_{30}$ . The parameter for  $lrelu$  was .2 for each case. All filters were size 4x4 with stride 2.

We placed the embedding into the following architecture to compute  $P(a|s_t) : FC_{64} \rightarrow lrelu \rightarrow FC_{64} \rightarrow FC_{|A|}$ . This is the same architecture as ILPO for  $P(a|s_t, z)$  except it does not take in latent actions. To compute inverse dynamics for  $s_t$  and  $s_{t+1}$ , we place each through  $E_p$  and concatenate them, then compute  $lrelu$  for the concatenation and place through  $FC_{|A|}$ .

We trained the network for 1000 iterations with a batch size of 100. At each iteration, we collected 200 steps of experience from the policy and trained the inverse dynamics model

for 500 steps and the policy for 500 steps. The learning rate was .0001. Training this model was very slow because it was done on images and needed to train the policy and inverse dynamics model at each iteration.

Both BCO and ILPO used  $\epsilon = .2$  for taking random actions while training and  $\epsilon = .1$  while evaluating.



## REFERENCES

- [1] ABBEEL, P. and NG, A. Y., “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*, p. 1, ACM, 2004.
- [2] AMMAR, H. B., EATON, E., RUVOLO, P., and TAYLOR, M. E., “Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [3] AMODEI, D., OLAH, C., STEINHARDT, J., CHRISTIANO, P., SCHULMAN, J., and MANÉ, D., “Concrete problems in ai safety,” *arXiv preprint arXiv:1606.06565*, 2016.
- [4] ANDRYCHOWICZ, M., WOLSKI, F., RAY, A., SCHNEIDER, J., FONG, R., WELINDER, P., MCGREW, B., TOBIN, J., ABBEEL, O. P., and ZAREMBA, W., “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.
- [5] ARGALL, B. D., CHERNOVA, S., VELOSO, M., and BROWNING, B., “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [6] AYTAR, Y., PFAFF, T., BUDDEN, D., PAINE, T. L., WANG, Z., and DE FREITAS, N., “Playing hard exploration games by watching youtube,” *arXiv preprint arXiv:1805.11592*, 2018.
- [7] BABES, M., MARIVATE, V., SUBRAMANIAN, K., and LITTMAN, M. L., “Apprenticeship learning about multiple intentions,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 897–904, 2011.
- [8] BARCZAK, A. L. C., REYES, N. H., ABASTILLAS, M., PICCIO, A., and SUSNJAK, T., “A new 2d static hand gesture colour image dataset for asl gestures,” *Research Letters in the Information and Mathematical Sciences*, vol. 15, pp. 12–20, 2011.
- [9] BHAT, S. and MATEAS, M., “On the difficulty of modular reinforcement learning for real-world partial programming,” 2006.
- [10] BISHOP, C. M. and NASRABADI, N. M., “Pattern recognition and machine learning,” *J. Electronic Imaging*, vol. 16, p. 049901, 2007.
- [11] BOBICK, A. F. and DAVIS, J. W., “The recognition of human movement using temporal templates,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 3, pp. 257–267, 2001.

- [12] BONIARDI, F., VALADA, A., BURGARD, W., and TIPALDI, G. D., “Autonomous indoor robot navigation using sketched maps and routes,” in *Workshop on Model Learning for Human-Robot Communication at Robotics: Science and Systems (RSS)*, Citeseer, 2016.
- [13] BOTTOU, L., “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, pp. 177–186, Springer, 2010.
- [14] CHEN, X., DUAN, Y., HOUTHOOFT, R., SCHULMAN, J., SUTSKEVER, I., and ABBEEL, P., “Infogan: Interpretable representation learning by information maximizing generative adversarial nets,” in *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.
- [15] CHENTANEZ, N., BARTO, A. G., and SINGH, S. P., “Intrinsically motivated reinforcement learning,” in *Advances in neural information processing systems*, pp. 1281–1288, 2005.
- [16] CHERNOVA, S. and THOMAZ, A. L., “Robot learning from human teachers,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.
- [17] CHIAPPA, S., RACANIÈRE, S., WIERSTRA, D., and MOHAMED, S., “Recurrent environment simulators,” *arXiv preprint arXiv:1704.02254*, 2017.
- [18] CHRISTIANO, P. F., LEIKE, J., BROWN, T., MARTIC, M., LEGG, S., and AMODEI, D., “Deep reinforcement learning from human preferences,” in *Advances in Neural Information Processing Systems*, pp. 4299–4307, 2017.
- [19] CLEVERT, D.-A., UNTERTHINER, T., and HOCHREITER, S., “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.
- [20] COBBE, K., KLIMOV, O., HESSE, C., KIM, T., and SCHULMAN, J., “Quantifying generalization in reinforcement learning,” *arXiv preprint arXiv:1812.02341*, 2018.
- [21] DALAL, N. and TRIGGS, B., “Histograms of oriented gradients for human detection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.
- [22] DAVIS, J., “Recognizing movement using motion histograms,” *Technical Report 487, MIT Media Lab*, vol. 1, no. 487, p. 1, 1999.
- [23] DEVIN, C., GUPTA, A., DARRELL, T., ABBEEL, P., and LEVINE, S., “Learning modular neural network policies for multi-task and multi-robot transfer,” *arXiv preprint arXiv:1609.07088*, 2016.
- [24] DHARIWAL, P., HESSE, C., KLIMOV, O., NICHOL, A., PLAPPERT, M., RADFORD, A., SCHULMAN, J., SIDOR, S., and WU, Y., “Openai baselines.” <https://github.com/openai/baselines>, 2017.

- [25] DIETTERICH, T. G., “Hierarchical reinforcement learning with the maxq value function decomposition,” *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.
- [26] EDWARDS, A., ISBELL, C., and TAKANISHI, A., “Perceptual reward functions,” *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI Workshop*, 2016.
- [27] EDWARDS, A. D., DOWNS, L., and DAVIDSON, J. C., “Forward-backward reinforcement learning,” *arXiv preprint arXiv:1803.10227*, 2018.
- [28] EDWARDS, A. D. and ISBELL JR, C. L., “Cross-domain perceptual reward functions,” *RLDM 2017*, 2017.
- [29] EDWARDS, A. D., SAHNI, H., SCHROECKER, Y., and ISBELL, C. L., “Imitating latent policies from observation,” *arXiv preprint arXiv:1805.07914*, 2018.
- [30] FINN, C., TAN, X. Y., DUAN, Y., DARRELL, T., LEVINE, S., and ABBEEL, P., “Deep spatial autoencoders for visuomotor learning,” *reconstruction*, vol. 117, no. 117, p. 240, 2015.
- [31] FINN, C., TAN, X. Y., DUAN, Y., DARRELL, T., LEVINE, S., and ABBEEL, P., “Deep spatial autoencoders for visuomotor learning,” in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 512–519, IEEE, 2016.
- [32] FINN, C., YU, T., FU, J., ABBEEL, P., and LEVINE, S., “Generalizing skills with semi-supervised reinforcement learning,” *arXiv preprint arXiv:1612.00429*, 2016.
- [33] GOODFELLOW, I., BENGIO, Y., and COURVILLE, A., *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [34] GOYAL, A., BRAKEL, P., FEDUS, W., LILICRAP, T., LEVINE, S., LAROCHELLE, H., and BENGIO, Y., “Recall traces: Backtracking models for efficient reinforcement learning,” *arXiv preprint arXiv:1804.00379*, 2018.
- [35] GRIFFITH, S., SUBRAMANIAN, K., SCHOLZ, J., ISBELL, C., and THOMAZ, A. L., “Policy shaping: Integrating human feedback with reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 2625–2633, 2013.
- [36] HARWATH, D., TORRALBA, A., and GLASS, J., “Unsupervised learning of spoken language with visual context,” in *Advances in Neural Information Processing Systems*, pp. 1858–1866, 2016.
- [37] HAUSMAN, K., CHEBOTAR, Y., SCHAAL, S., SUKHATME, G., and LIM, J. J., “Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets,” in *Advances in Neural Information Processing Systems*, pp. 1235–1245, 2017.
- [38] HO, J. and ERMON, S., “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

- [39] HUTCHINSON, S., HAGER, G. D., CORKE, P., and OTHERS, “A tutorial on visual servo control,” *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 5, pp. 651–670, 1996.
- [40] IOFFE, S. and SZEGEDY, C., “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [41] IRANI, A. J., *Utilizing negative policy information to accelerate reinforcement learning*. PhD thesis, Georgia Institute of Technology, 2015.
- [42] ISBELL, C., SHELTON, C. R., KEARNS, M., SINGH, S., and STONE, P., “A social reinforcement learning agent,” in *Proceedings of the fifth international conference on Autonomous agents*, pp. 377–384, ACM, 2001.
- [43] KANADE, T., COHN, J. F., and TIAN, Y., “Comprehensive database for facial expression analysis,” in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pp. 46–53, IEEE, 2000.
- [44] KAPLAN, R., SAUER, C., and SOSA, A., “Beating atari with natural language guided reinforcement learning,” *arXiv preprint arXiv:1704.05539*, 2017.
- [45] KILNER, J. M. and FRITH, C. D., “Action observation: inferring intentions without mirror neurons,” *Current Biology*, vol. 18, no. 1, pp. R32–R33, 2008.
- [46] KINGMA, D. and BA, J., “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [47] KIRK, J., MININGER, A., and LAIRD, J., “Learning task goals interactively with visual demonstrations,” *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 1–8, 2016.
- [48] KISHI, T., OTANI, T., ENDO, N., KRYCZKA, P., HASHIMOTO, K., NAKATA, K., and TAKANISHI, A., “Development of expressive robotic head for bipedal humanoid robot,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4584–4589, IEEE, 2012.
- [49] KNOX, W. B. and STONE, P., “Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance,” *Artificial Intelligence*, vol. 225, pp. 24–50, 2015.
- [50] LI, Y., SONG, J., and ERMON, S., “Infogail: Interpretable imitation learning from visual demonstrations,” in *Advances in Neural Information Processing Systems*, pp. 3815–3825, 2017.
- [51] LIU, Y., GUPTA, A., ABBEEL, P., and LEVINE, S., “Imitation from observation: Learning to imitate behaviors from raw video via context translation,” *arXiv preprint arXiv:1707.03374*, 2017.

- [52] LUCEY, P., COHN, J. F., KANADE, T., SARAGIH, J., AMBADAR, Z., and MATTHEWS, I., “The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression,” in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pp. 94–101, IEEE, 2010.
- [53] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., and RIEDMILLER, M., “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [54] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., and OTHERS, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [55] NAIR, A., MCGREW, B., ANDRYCHOWICZ, M., ZAREMBA, W., and ABBEEL, P., “Overcoming exploration in reinforcement learning with demonstrations,” *arXiv preprint arXiv:1709.10089*, 2017.
- [56] NG, A. Y., HARADA, D., and RUSSELL, S., “Policy invariance under reward transformations: Theory and application to reward shaping,” in *ICML*, vol. 99, pp. 278–287, 1999.
- [57] NG, A. A. Y. and RUSSELL, S. J. S., “Algorithms for inverse reinforcement learning,” in *Icml*, vol. 0, pp. 663–670, 2000.
- [58] OH, J., GUO, X., LEE, H., LEWIS, R. L., and SINGH, S., “Action-conditional video prediction using deep networks in atari games,” in *Advances in Neural Information Processing Systems*, pp. 2863–2871, 2015.
- [59] PATHAK, D., MAHMOUDIEH, P., LUO, G., AGRAWAL, P., CHEN, D., SHENTU, Y., SHELHAMER, E., MALIK, J., EFROS, A. A., and DARRELL, T., “Zero-shot visual imitation,” *arXiv preprint arXiv:1804.08606*, 2018.
- [60] POMERLEAU, D. A., “Alvin: An autonomous land vehicle in a neural network,” in *Advances in neural information processing systems*, pp. 305–313, 1989.
- [61] QURESHI, A. H., NAKAMURA, Y., YOSHIKAWA, Y., and ISHIGURO, H., “Robot gains social intelligence through multimodal deep reinforcement learning,” in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pp. 745–751, IEEE, 2016.
- [62] RIZZOLATTI, G. and SINIGAGLIA, C., “The functional role of the parieto-frontal mirror circuit: interpretations and misinterpretations,” *Nature reviews neuroscience*, vol. 11, no. 4, p. 264, 2010.
- [63] RUSU, A. A., VECERIK, M., ROTHÖRL, T., HEES, N., PASCANU, R., and HADSELL, R., “Sim-to-real robot learning from pixels with progressive nets,” *arXiv preprint arXiv:1610.04286*, 2016.

- [64] SADEGHI, F. and LEVINE, S., “(cad)2rl: Real single-image flight without a single real image,” *arXiv preprint arXiv:1611.04201*, 2016.
- [65] SALDIEN, J., GORIS, K., VANDERBORGHT, B., VANDERFAEILLIE, J., and LEFEBER, D., “Expressing emotions with the social robot probot,” *International Journal of Social Robotics*, vol. 2, no. 4, pp. 377–389, 2010.
- [66] SCHAAL, S., “Robot learning from demonstration,” *Neural Information Processing System (NIPS)*, 1997.
- [67] SCHAAL, S., “Is imitation learning the route to humanoid robots?,” *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.
- [68] SCHAUL, T., HORGAN, D., GREGOR, K., and SILVER, D., “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1312–1320, 2015.
- [69] SCHROECKER, Y. and ISBELL, C. L., “State Aware Imitation Learning,” *Advances in Neural Information Processing Systems 30*, no. Nips, pp. 2915–2924, 2017.
- [70] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., and KLIMOV, O., “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [71] SERMANET, P., LYNCH, C., HSU, J., and LEVINE, S., “Time-contrastive networks: Self-supervised learning from multi-view observation,” *arXiv preprint arXiv:1704.06888*, 2017.
- [72] SERMANET, P., XU, K., and LEVINE, S., “Unsupervised perceptual rewards for imitation learning,” *arXiv preprint arXiv:1612.06699*, 2016.
- [73] SHAH, D., SCHNEIDER, J., and CAMPBELL, M., “A sketch interface for robust and natural robot control,” *Proceedings of the IEEE*, vol. 100, no. 3, pp. 604–622, 2012.
- [74] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., SANDER, D., GREWE, D., NHAM, J., KALCHBRENNER, N., SUTSKEVER, I., LILLICRAP, T., LEACH, M., KAVUKCUOGLU, K., GRAEPEL, T., and HASSABIS, D., “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [75] SINGH, S., LEWIS, R. L., and BARTO, A. G., “Where do rewards come from,” in *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606, 2009.
- [76] SKUBIC, M., BLISARD, S., CARLE, A., and MATSAKIS, P., “Hand-drawn maps for robot navigation,” in *AAAI Spring Symposium, Sketch Understanding Session*, p. 23, 2002.

- [77] SKUBIC, M., MATSAKIS, P., FORRESTER, B., and CHRONIS, G., “Extracting navigation states from a hand-drawn map,” in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 1, pp. 259–264, IEEE, 2001.
- [78] STADIE, B. C., ABBEEL, P., and SUTSKEVER, I., “Third-person imitation learning,” *arXiv preprint arXiv:1703.01703*, 2017.
- [79] STOLLE, M. and PRECUP, D., “Learning options in reinforcement learning,” in *International Symposium on abstraction, reformulation, and approximation*, pp. 212–223, Springer, 2002.
- [80] STRAMANDINOLI, F., CANGELOSI, A., and MAROCCO, D., “Towards the grounding of abstract words: a neural network model for cognitive robots,” in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pp. 467–474, IEEE, 2011.
- [81] SUBRAMANIAN, K., ISBELL, C. L., and THOMAZ, A. L., “Learning options through human interaction,” in *In 2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT)*, Citeseer, 2011.
- [82] SUTTON, R. S., “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming,” in *Proceedings of the seventh international conference on machine learning*, pp. 216–224, 1990.
- [83] SUTTON, R. S. and BARTO, A. G., *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.
- [84] SUTTON, R. S. and BARTO, A. G., *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.
- [85] TAYLOR, M. E. and STONE, P., “Transfer learning for reinforcement learning domains: A survey,” *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.
- [86] TENORTH, M., KLANK, U., PANGERCIC, D., and BEETZ, M., “Web-enabled robots,” *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 58–68, 2011.
- [87] THOMAZ, A. L. and BREAZEAL, C., “Teachable robots: Understanding human teaching behavior to build more effective robot learners,” *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.
- [88] TOBIN, J., FONG, R., RAY, A., SCHNEIDER, J., ZAREMBA, W., and ABBEEL, P., “Domain randomization for transferring deep neural networks from simulation to the real world,” *arXiv preprint arXiv:1703.06907*, 2017.
- [89] TORABI, F., WARNELL, G., and STONE, P., “Behavioral cloning from observation,” *arXiv preprint arXiv:1805.01954*, 2018.

- [90] TROVATO, G., KISHI, T., ENDO, N., HASHIMOTO, K., and TAKANISHI, A., “Development of facial expressions generator for emotion expressive humanoid robot,” in *Humanoid Robots (Humanoids)*, 2012 12th IEEE-RAS International Conference on, pp. 303–308, IEEE, 2012.
- [91] TZENG, E., DEVIN, C., HOFFMAN, J., FINN, C., ABBEEL, P., LEVINE, S., SAENKO, K., and DARRELL, T., “Adapting deep visuomotor representations with weak pairwise constraints,” in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
- [92] VEERIAH, V., PILARSKI, P. M., and SUTTON, R. S., “Face valuing: Training user interfaces with facial expressions and reinforcement learning,” *arXiv preprint arXiv:1606.02807*, 2016.
- [93] WIEWIORA, E., “Potential-based shaping and q-value initialization are equivalent,” *Journal of Artificial Intelligence Research*, vol. 19, pp. 205–208, 2003.
- [94] YANG, Y., LI, Y., FERMULLER, C., and ALOIMONOS, Y., “Robot learning manipulation action plans by “Watching” unconstrained videos from world wide web,” in *AAAI 2015*, (Austin, US), jan 2015.
- [95] ZHANG, F., LEITNER, J., MILFORD, M., UPCROFT, B., and CORKE, P., “Towards vision-based deep reinforcement learning for robotic motion control,” *arXiv preprint arXiv:1511.03791*, 2015.
- [96] ZHU, J.-Y., ZHANG, R., PATHAK, D., DARRELL, T., EFROS, A. A., WANG, O., and SHECHTMAN, E., “Toward multimodal image-to-image translation,” in *Advances in Neural Information Processing Systems*, pp. 465–476, 2017.
- [97] ZHU, Y., MOTTAGHI, R., KOLVE, E., LIM, J. J., GUPTA, A., FEI-FEI, L., and FARHADI, A., “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” *arXiv preprint arXiv:1609.05143*, 2016.
- [98] ZOPH, B. and LE, Q. V., “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.
- [99] ZUO, X., “mazelab: A customizable framework to create maze and gridworld environments..” <https://github.com/zuoxingdong/mazelab>, 2018.